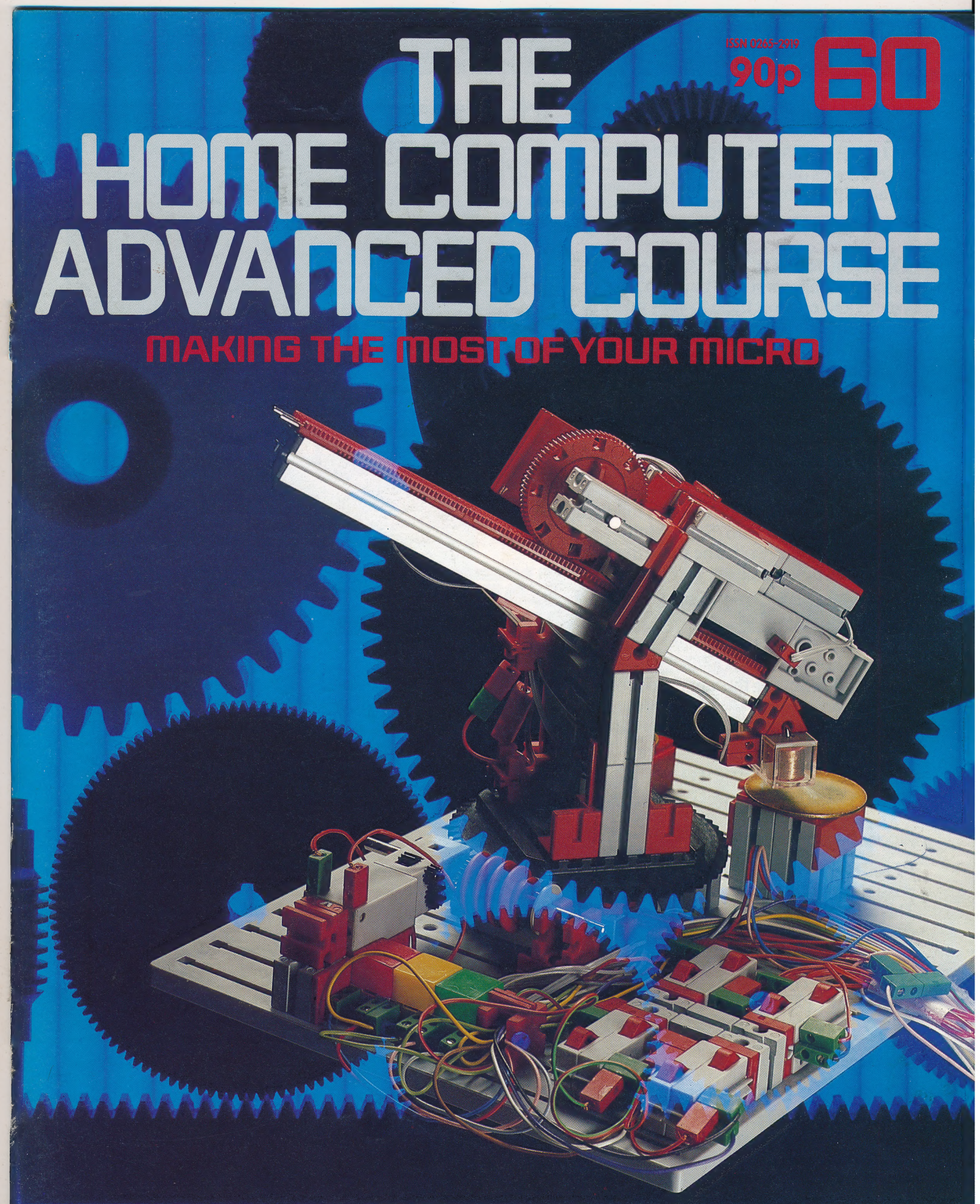
ISSN 0265-2919

90p 60

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

FUTURE FORM As we conclude the present series on education, we turn our attention to the future — 1181

HARDWARE

SIX-PACK ROBOT The Fischertechnik Robotics kit allows you to build up your own systems and connect them to your micro. We examine its potential — 1189

SOFTWARE

DOUBLE-ENTRY DATABASE The power of a good database manager is clearly seen when information from one file needs relating to data from another. We provide specific examples — 1186

BAD MOON RISING Ultimate's follow-up to Sabre Wulf has a hero (Sabre Man) turning into a werewolf. The game displays some of the best Spectrum graphics yet seen — 1200

COMPUTER SCIENCE

VALUABLE VARIABLES We look at PASCAL's use of functions and some sample programs illustrating the language's concise structure — 1184

JARGON

FROM OPTICAL DISC TO ORDERED PAIR A weekly glossary of computing terms — 1188

PROGRAMMING PROJECTS

PLAGUED WITH PROBLEMS Six possible major events can affect your voyage. This week's module deals with a stranded crew and a visitation by the plague — 1192

MACHINE CODE

SIGHT EFFECTS We look at the VIC-II video controller chip for the Commodore 64 and see how its registers create different display modes and visual effects — 1198

WORKSHOP

ELBOW GREASE We continue the programming of the robot arm by providing the software for the Commodore 64 — 1195

INDEX A complete index to issues 49 to 60 — BACK COVERS

Next Week

- **In our Commodore 64 operating system series we look at a program that displays high and low resolution graphics to be displayed simultaneously**
- **We take a look at computerised biofeedback systems, which among uther things allow 'joystick' control to be performed by electrodes attached to the temples**
- **The addition of a disk drive to the Amstrad range enables the computer to be used as a powerful business machine. We home in on the drive and the software included**

QUIZ

1) Why is it necessary to blank out the screen when we run the robot arm from the Commodore 64?
2) Why is the Prof command system inappropriate for driving the Fischertechnik graphics input device?
3) What is a CD-ROM?
4) What is the storage capacity of an optical disk?

Answers To Last Week's Quiz

1) The 6303 processor is fitted to enable the Penman to use the various languages and operating systems available.
2) A relational database manager requires that each item has a separate record of a fixed length, whereas a hierarchical database manager is organised in smaller segments and fields.
3) A 'parameter list' is used to assign local variables within a PASCAL procedure.

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Deputy Editor** Steve Colwill; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Steve Malone; **Sub Editor** Jon Kaye; **Contributors** Dr Andrew Bangham, Nick Walsh, Dr Antonia Jones, Steve Malone, David Mudd, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985; Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited. Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CHRIS STEVENS

FUTURE FORM

Imminent developments in hardware and a revolution in data storage techniques look set to make the traditional classroom gathering an irrelevance for both teacher and pupil. We conclude our present series on education with a conjectural assessment of the way in which computer technology may change educational methods.

The future of educational computing presents a sharp contrast to its past. Cheaper hardware, improved software and technological development are all combining to change the face of education as we know it. Far from being the poor relation of the consumer market, education is rapidly establishing itself as one of the major growth areas in the development of a new technology. The reason for this sudden reversal in the image of educational computing is not hard to pinpoint. In the early days of the technology revolution, education was consistently overlooked by hardware and software companies in favour of the more lucrative consumer market. This situation is changing rapidly as producers realise the importance of selling into schools and promoting brand awareness among children who will later be consumers in their own right.

The immediate effect of this is cheaper and more readily available hardware. In 1985 Apple began a campaign selling the Macintosh into schools at a 30 per cent discount. Atari has followed Apple by designing its new range of ST 32-bit micros with the education market in mind,

ALAN ADLER

Hub Of Learning
Computers may revolutionise our ideas of education by placing the child at the centre of the learning process, in control of the different teaching resources that new technology will make available. Satellite technology and networking systems will enable the student to communicate with knowledge databases from different countries and cultures around the world. Schools may give way to 'learning supervision centres', where children meet to discuss their work, most of which could be completed at home. Interactive video systems can form a basis for the development of highly complex software and robots may be used in physical demonstrations of abstract concepts. Additionally, local mainframe systems may be accessed by the student, using a micro as an intelligent terminal, and such systems could supervise exchange of data, pupil-to-pupil communication, and offer increased processing power when necessary

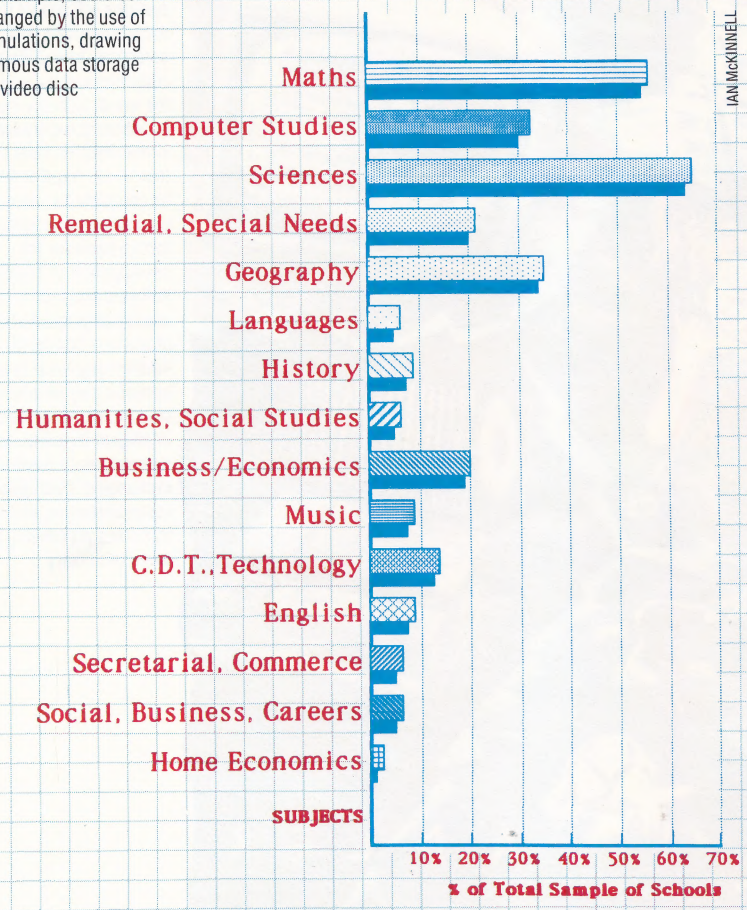
providing the user with the option of purchasing a machine with LOGO, rather than BASIC, supplied in ROM. In the UK, Amstrad, Acorn, Sinclair and RML are all competing for a foothold in schools, with Amstrad, for example, aiming for a sale of 100 units per day to the education market.

Apart from the commercial sense of penetrating the education market, there has also arisen a second generation of computer users who demand more serious uses for their micros. Software development that might previously have been targeted solely at schools is suddenly finding a wider market, making investment in 'serious' educational software a better commercial proposition. Lack of hardware and little software development are therefore no longer such a hindrance to educational computing, so what does the future hold? The use of computers in education can now proceed in two directions: the extension and improvement of existing techniques, and the introduction of entirely new methods.

Seymour Papert has already indicated a number of extensions he would like to make to the LOGO concept using the power of machines like the Macintosh. Papert wants to see a 'physics microworld' written into the language, allowing children to play with the laws of motion the same way they play with the laws of geometry. He also foresees a 'database microworld' in which children learn about information processing. Some of Papert's ideas of the physics microworld have already been incorporated into Mac'LOGO.

Video Verisimilitude

Departmental use of micros in schools shows a heavy bias towards scientific applications. The introduction of interactive video systems and the development of IV software could well do much to redress this balance. The teaching of history, for example, could be radically changed by the use of complex simulations, drawing on the enormous data storage potential of video disc technology



Simulations, CAL programs and databases can all become more friendly and more powerful as programs like the Macintosh Filevision find their way into schools.

The most exciting areas of educational computing, however, revolve around the introduction of entirely new technologies, in particular the CD-ROM, or compact disc read-only memory, and interactive video (IV). The possibility of vast amounts of data storage at a reasonable price enables educational software to break free of the 'linear' approach typical of programs developed on small, unfriendly systems. The degree of interaction in a 48 Kbyte CAL program is extemely limited and such software necessarily imposes restrictions on the learning process. With IV the programmer can allow the student far greater flexibility in approaching a topic, making use of the increased memory space to allow for different directions to be taken during program execution.

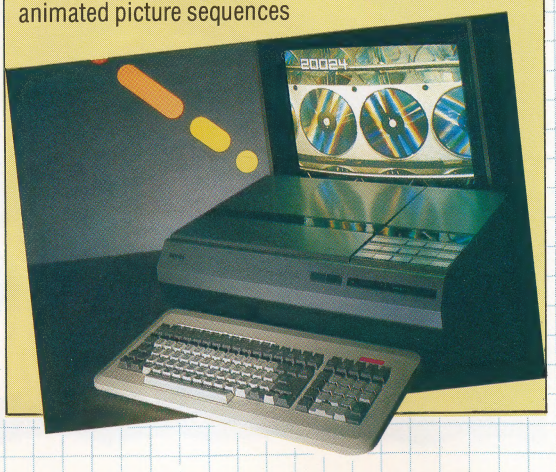
A recent Council for Educational Technology working party identified four areas where IV could be of use in schools. Teachers could use it as a teaching aid, providing instructional programs and audiovisual references. Pictures and sequences could be rapidly accessed. Small groups of children could use it as a 'tutor', discussing the

Domesday Plus 900

The enormous storage capacity of the video disc opens up entirely new applications possibilities. The BBC, in conjunction with Philips, Thorn-EMI and the Microcomputers in Education Project, is commemorating the 900th anniversary of the Domesday Book by setting up a nationwide project to gather data on contemporary Britain.

Schools all over the country will collect data relating to their immediate environment, all of which will be collated to form a 'people's database' containing details of land use, amenities, services, etc. Additionally, national data sources (for example, the Census and transport surveys) will be combined into a vast 'national database', and both databases will reside on a single laser disc.

The Domesday Project will present a remarkable record of contemporary society, and thanks to laser disc technology the record will be able to contain not only text but also maps, diagrams, tables and even animated picture sequences



material and answering questions. Individuals could use it for full interactive simulations, sophisticated CAL programs and open-ended learning. Finally, it could be used for data access, as a library of audio and video reference materials.

Although much work needs to be done in indexing and accessing information and developing software for IV, the system is already being introduced in many different areas. The Open University has used it in summer schools where, because of the short time available, engineering students have difficulty in studying the effect of, for example, stress on various materials. There has been an IV program written that uses still frames, animation and video sequences. It takes the form of a courtroom battle between the manufacturers of teddy bears and the suppliers of washers, used to secure the bears' eyes. The student must conduct a series of experiments, which will form the basis of the evidence used in court. The interactive program is controlled by a large CAL program which 'discusses' possible causes of polymer failure in the substance used in the bears' eyes. The student performs various experiments and interprets the results, which are finally presented in a dramatised sequence.

NO MORE SCHOOLS

The transfer of existing software techniques — simulation, CAL, database and so on — onto CD-ROM systems is exciting, but the long-term effects of the new technology go much farther. 'I do not believe,' says Seymour Papert, 'that there will be schools in 20 years' time. It is impossible that the concept of a teacher in front of a class taking notes can exist in the age of computers. If you go one by one down through all the things which happen in schools they just will not be needed, except for the things that happen very badly in school, like children talking to one another and getting to know one another . . . Something will exist where children meet and learn and develop. Whether we call it school or not is not very important. What is important is that school as we know it today will not exist.'

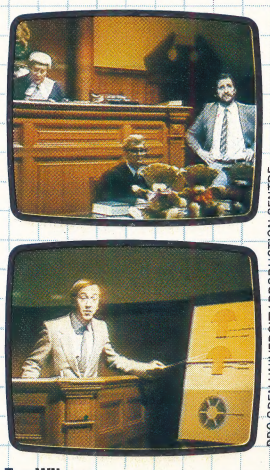
His views are echoed, and taken further, by Professor Tom Stonier: 'The computer will initiate the shift from school based education to home based education. The teacher's role will change from walking database to knowledge counsellor.'

The possible implications of this shift in emphasis from a system-oriented educational structure to one centred around the individual student are enormous and by no means confined to the process of learning. Professor Stonier has co-authored a book entitled *Children, Computers and Communications* in which he predicts education by the 'electronic grandmother' system.

According to Stonier, as education shifts from school to home and improvements in healthcare and medicine increase the average lifespan, senior citizens will become more involved in education, as indeed they already are in many so-called 'primitive' societies. Stonier believes that the aged, one of our most wasted resources, are ideally qualified to take responsibility for a section of the education system. An example of how the system might develop already exists in America. In Portage, Wisconsin, a teacher visits families for half a day each week. During the visit, the teacher assesses progress made by the child during the previous week and decides, in collaboration with the parents, on the direction of next week's work.

Although the Portage system centres around the education of handicapped children, there is every possibility of extending it to other students. In Britain it is legal to educate your children at home if you can convince the local education authority that you are capable of doing so. An increasing number of families are opting out of the system and there are presently over 2000 who have 'deschooled' their children. Access to vast public databases, powerful micros with vast CD-ROM databases, and interactive software will give parents a powerful tool for educating their children at home.

As the complexity and scope of educational software increases, the role of the teacher becomes less and less easy to define. Stonier predicts that students will become more knowledgeable than their teachers. As long as teachers fulfil the function of 'walking databases', the knowledge a child acquires is severely limited by the teacher's understanding. Every teacher of computer studies, and indeed most subjects, can point to children who have overtaken them. Stonier believes that this pattern will be repeated across the curriculum, as databases and educational software develop. Teachers will fulfil the role of 'knowledge counsellors'. They will become experts on learning, rather than on particular subjects, guiding and directing the children in their studies, rather than 'teaching' them.



BBC OPEN UNIVERSITY PRODUCTION CENTRE

Eye Witness
Animated film sequences and other high-resolution visual information involve the processing and storage of vast amounts of data — hitherto impossible on microcomputer-controlled systems but now rapidly becoming available and affordable using interactive video (IV) technology. These screen shots from 'The Teddy Bear Disc' show a computer-controlled video disc system in action, in this case as a tutorial for undergraduates in engineering materials. The program is set in an industrial court and IV technology enables the student to play the part of the 'expert witness', carrying out experiments and using the system to construct a legal case

CD-ROM Revolution

To store data reliably, much of the space on CD-ROMs has to be dedicated to error-checking routines and formatting information. Nevertheless, one CD-ROM disc offers the user 552 megabytes of data storage, enabling thousands of screens of information to be stored on one disc.

Furthermore, unlike audio CD disc players, which need sophisticated D/A (digital/analogue) converters, CD-ROM players have to deal only with digital data, and this simplifies the player circuitry and reduces unit cost. Bearing in mind the players are otherwise almost identical to audio units, prices are likely to be kept low by consumer demand and standard manufacturing techniques.

Audio compact discs sell for around £10, so CD-ROM discs should sell for similar amounts if sold in large enough quantities. Since a single CD-ROM disc could easily hold all the applications software you are ever likely to need, a big shake-up in software marketing will almost certainly follow their introduction

VALUABLE VARIABLES

As we saw in our previous instalment, PASCAL allows no such thing as a function statement — functions compute values rather than invoke procedures for them. Here, we look at the use of functions within parameters and provide several examples that illustrate PASCAL's strict, and therefore concise, structural integrity.

Just as procedures may be defined and called by name, functions may also be called, but only in expressions — not as statements. Whereas procedure calls invoke the execution of subprograms to perform some process, function calls *compute* a value. The result returned may be of any simple type, real or scalar, or a 'pointer' type (which we have yet to deal with). The only difference in the heading, besides using the reserved word FUNCTION, is that the type identifier of the returned result must be specified. Suppose that PASCAL did not have the predefined function odd — we could easily define our own:

```
FUNCTION Odd (number : integer) : boolean;
  BEGIN
    Odd := number MOD 2>>0
  END; {Odd}
```

The value is returned in the function identifier, and therefore must be assigned to it in the body of the function, as shown. In this respect, function names are like variables which are never initialised, but their values are computed whenever they appear in an expression. They can only appear on the LHS (left-hand side) of an assignment, so if we were careless (and long-winded) and programmed the assignment as:

```
IF number MOD 2 > 0 THEN
  Odd := true
```

(forgetting the necessary clause ELSE Odd := false), a path through the construct exists leaving the result undefined.

As with the procedure parameters we have considered so far, the value of the actual parameter is passed to the local integer identifier, number, from the 'activation' point. So:

```
WriteLn ( Odd ( sqr ( N DIV 100 ) ) )
```

which would always print False, performs four operations :

1. the expression N DIV 100 is calculated
2. this temporary integer value is passed to sqr as an actual value parameter
3. its square is returned and passed to Odd
4. the Boolean result is now evaluated and passed to the WriteLn procedure as another value parameter

If N had the value 17 when the statement above was executed, what would its value be afterwards? This may seem like a ridiculous question, of course. There is no possible reason for N's value to be changed during evaluation of any function involving N as a parameter. The results of all functions should merely depend on the 'state of the world' as it exists. That is, the returned value is a 'function' (sic) of the value(s) of their 'arguments' or parameters. PASCAL's mechanism for passing only the value of variables ensures that even if the parameters are changed locally by a function or procedure, and because they are in fact local copies, the actual parameters will not be altered at the activation point.

GLOBAL DATA VS PARAMETERS

For these same reasons, although PASCAL does not forbid it, data should not be accessed globally. All data linkage between procedure and function calls should be controlled by parameter lists — even though the data is within the subprogram's scope. The only exception to this general rule is for the access of global constants. Because of their very nature, no harm can befall them in PASCAL, as it is impossible to alter their values. If a constant value were passed as a parameter, however, it would become a local variable and therefore no longer secure.

```
FUNCTION Lower (character :char)                :char;
  {returns the lower case of any upper case
  argument }
  CONST
    offset = 32; { ASCII ord ('a') — ord ('A')}
  BEGIN
  IF character IN [ 'A' . . 'Z' ]
    THEN
      Lower := chr ( ord (character ) + offset )
    ELSE
      Lower := character
  END; { Lower }
```

When we are defining procedures, it is sometimes essential that they *should* alter the values of their parameters — otherwise they would not perform the task for which they were designed. An obvious example is PASCAL's own read procedure. It would not be very helpful if read (N) merely gave a value to an integer that was local to read and the value of N were unchanged. In such cases, we need to pass the address of a variable parameter (rather than its value) in order that the procedure may refer directly to it rather than a local copy. This mechanism is called 'passing by address' or 'by reference', and should normally be used only with procedures, not functions.

Passing of variable parameters is achieved simply by including the reserved word VAR before any item (or items) in the procedure's parameter list. The syntax of a parameter list can be seen to be identical to the VAR declaration of a block; but instead of VAR appearing once as a delimiter, it

only appears before an item that needs to be VARied by a procedure. For instance:

```
PROCEDURE Process ( VAR counter : CountList );
```

APPLICATION DEVELOPMENT

Let's illustrate this powerful technique by developing a program to read some names from the keyboard and, associated with each name, an amount of money that each person owes us. For simplicity, we'll just use one string per name and express the debts as integer amounts in pounds sterling. Once the program is tested, we could add further details such as addresses, telephone numbers and so on. We would like to print out the list of names either in ascending alphabetical order or possibly in order of largest amounts owing. The natural data structure will be a list of records containing fields of the appropriate type that can easily be added to later. We can handle each record as a single piece of data, but use any of the fields we choose as a 'key' for sorting. (For more information on 'keys' see page 1124.) First, let's design the full data representation:

```
CONST
   StringLength = 20;
   ListLength   = 50;
TYPE
   Cardinal     = 0 . . MaxInt;
   StringSize   =1 . . StringLength;
   string       = PACKED ARRAY  [ StringSize ]
                                OF char;
   data         = RECORD
                        name : string;
{other fields to suit}  debt : Cardinal;
                  END;{data}
   bounds       =1 . . ListLength;
   RecordList   = ARRAY [ bounds ] OF data;
```

This allows for fifty names and assumes (for now) that no name will exceed twenty characters, but by using definitions we can easily alter this.

We have expressed all the subranges and structures in a TYPE definition part for several reasons:

1. security — an index of type bounds can never exceed the defined limits of the array, for example
2. tracing bugs — should such a 'range error' occur during testing, the error message will help locate the trouble
3. convenience and efficiency — local variables can have existing types, avoiding duplication and saving memory
4. necessity — PASCAL insists that parameter type 'denoters' must be identifiers, not 'literals' (0 . . 255, for example)

Next, we have to think about a few essential variables and an algorithm, all of which will be discussed in the next part of the series. In the meantime, try to develop your own 'informal algorithm' for the program and make a list of any procedures that you feel may be necessary.

Positive Advantage

The program CardReader handles positive numbers in a fully 'bomb-proof' way by reading the data as characters. Conversion to the equivalent numeric value is performed simply by subtracting the original value of the digit character from that of the digit '0'. This will work for all character sets, as the characters '0' . . . '9' are defined to be contiguous. Notice that the program has only two global variables (at 'level 0').

The data for processing the numbers and the function Value which performs the mapping is used only by ReadCard, and therefore all this is declared and defined within this procedure.

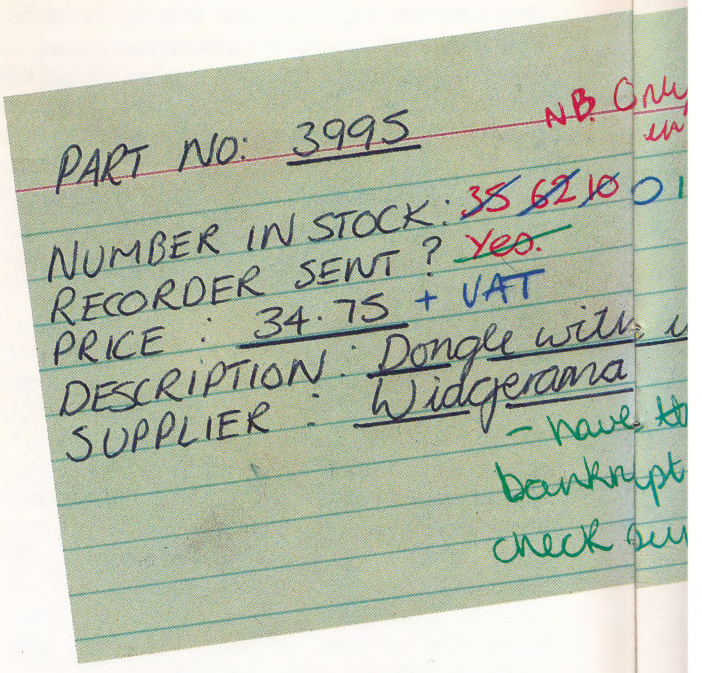
There is only one bug in the program as it stands — if a value greater than the cardinality of the compiler's integers (MaxInt) is entered, the assignment statement in the WHILE loop will crash.

What about writing another procedure ReadInt to cope with negative numbers?

```
PROGRAM        CardReader   ( input, output );

TYPE
       cardinal          = 0 .. MaxInt;

VAR
       PosNum   : cardinal;
       okay     : boolean;

       {11111111111111111111111111111111}

PROCEDURE   ReadCard  ( VAR   N    : cardinal;
                        VAR   OK   : boolean );
  CONST
       space    = ' ';
  TYPE
       single   = 0 .. 9;
  VAR
       symbol   : char;
       digits   : SET OF char;
         {22222222222222222222222}
                 { level 2 }
  FUNCTION  Value ( digit  : char )   : single;
       { returns the numeric value of
       a digit on any character set }
    BEGIN
      Value := ord ( digit ) - ord ( '0' )

    END;     { Value }
             {22222222222222222222222}
                    { back to level 1 }
BEGIN { ReadCard }

  REPEAT
    read ( symbol )

  UNTIL symbol > space;

  digits := [ '0' .. '9' ];

  OK := symbol IN digits;

  IF OK THEN
    BEGIN
      N := 0;

      WHILE  symbol IN digits  DO
        BEGIN { compute to base 10 }
          N := 10 * N + Value ( symbol );
          read ( symbol )
        END { any delimiting char or 'EoLn' }
    END     { after the number is discarded }

END;  { ReadCard }
      {11111111111111111111111111111111}
BEGIN   { CardReader - Level 0 Main program }

  page ( output );
  WriteLn;
  WriteLn ( '  This program reads positive' );
  WriteLn ( 'integers (in the range 0 .. ',
                MaxInt : 1, ')' );
  WriteLn ( 'Termination of the program is' );
  WriteLn ( 'controlled by error detection.' );
  WriteLn;
  write ( 'Enter a number : ' );
  ReadCard ( PosNum, okay );

  WHILE okay DO
     BEGIN
       WriteLn;
       write ( 'The number entered was : ' );
       WriteLn ( PosNum : 1 );
       write ( 'Number ? ' );
       ReadCard ( PosNum, okay )
     END;

  WriteLn ( '---  ERROR DETECTED  ---' : 40 )
END .
```

DOUBLE-ENTRY DATABASE

It is important to recognise when considering DBMs that certain types of files require different approaches to the organising, cross-referencing and retrieval of data. We present a number of examples illustrating these different approaches, from the straightforward 'flat' database to the complex and powerful dBase II.



Any record held in a 'flat' database will be completely self-contained; that is, any given record will hold all the information required, without the need to refer to another record. Traditional card-box files are flat in this sense. We can think of a library database, for example: each record consists simply of the fields TITLE, AUTHOR, PUBLISHER and ISBN (International Standard Book Number), and no cross-referencing is required.

Sometimes a field will contain an entry that can have a record in its own right. In a club membership file, for example, a CHILDREN field can refer to other records in the same file; if the children of a member are also club members they will have records of their own. But a field can refer to records that might not fit into the general structure of the database file. Consider a parts inventory database, containing the fields PART NUMBER, PRICE, NUMBER IN STOCK and SUPPLIER. It's highly likely in this case that the SUPPLIER field will also refer to other records that do not fit into the structure defined for the PARTS database.

To illustrate this, here, first of all, is the CLUB MEMBERSHIP file:

MEMBER'S NAME	Sue Gomez
YEAR JOINED	1979
SUBSCRIPTION?	Paid
OCCUPATION	Teacher
SALARY	8900
CHILDREN	Jill Gomez, Philip Gomez

In this example, the field CHILDREN contains two entries; both of these could have similar entries in the CLUB MEMBERSHIP file, if they were also members. Compare this with a PARTS database record:

PART NUMBER	3995
NUMBER IN STOCK	86
PRICE	34.75
DESCRIPTION	Dongle with widget nozzle
SUPPLIER	Widgerama Ltd; Dongle Corp of Taiwan

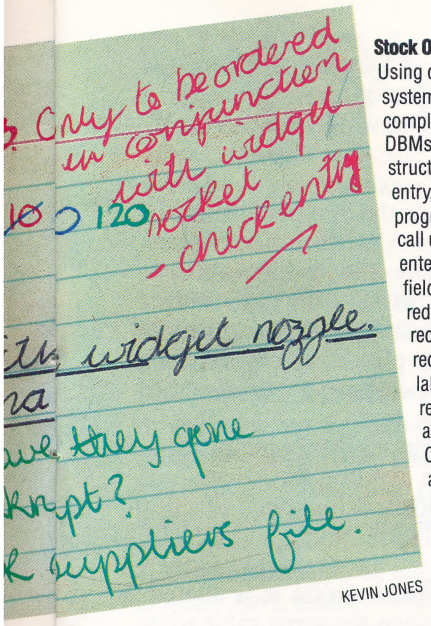
There are two entries in the SUPPLIER field; neither of these could have records that could fit into the structure of the PARTS database. So, a parts inventory such as this would probably have a second database file for suppliers, in which each entry might look something like this:

SUPPLIER	Dongle Corp of Taiwan
ADDRESS	57 Kau Moo Road, Taipei, Taiwan, R.O.C.
TELEPHONE	010-886-2-223-4478
SUPPLIES1	Dongle with widget nozzle
PRICE1($US)	34.75
SUPPLIES2	Valve paste (½ litre can)
PRICE2($US)	6.00
SUPPLIES3	Dazzlebrite polish (small can)
PRICE3($US)	2.30
SUPPLIES4	Dazzlebrite polish (large can)
PRICE4($US)	4.00
SUPPLIES5	—

It is perfectly obvious that the information we need about each of the suppliers is quite different in structure from the information we need about the parts the company sells. The solution is to have two different database files — one for the parts themselves, and another, separate file, for each of the suppliers.

If separate but related files are to be usefully implemented, it is obviously necessary for the DBM to be able to handle more than one file at a time. Less sophisticated DBMs, such as Card Box (see page 1104), can work only on a single file at a time, but DBMs such as dBase II (see page 1152) can use one file (such as PARTS) as a primary file and another (such as SUPPLIERS) as a secondary file. A really good DBM will allow a sort to be run on a primary file (to extract selected records) as well as the extraction of relevant records (such as SUPPLIERS) from related files.

The dBase II package allows two related files to be used at the same time, and associates them by using common key fields. Two commands allow reference to be switched from one to the other:



KEVIN JONES

Stock Order
Using our own index card filing system, entries can be as complex (or untidy) as we wish. DBMs have to adopt a more structured approach to data entry, but sophisticated programs will allow the user to call up related files and, when entering data, to skip certain fields if they have been made redundant by other entries in the record. Taking the illustrated record as an example, the field labelled 'Re-order sent' is relevant only if the stock approaches or falls to zero. Consequently some DBMs, such as 'Rescue', enable the user to set up fields to be dependent on, or calculated by, previous fields

SELECT PRIMARY and SELECT SECONDARY. If the primary file is PARTS and the secondary file is SUPPLIER, and we were working on the PARTS file, we would issue the command USE PARTS. If we then wanted to cross-refer to the suppliers file we would issue the commands: SELECT SECONDARY, and then on the next line, USE SUPPLIER. All the usual dBase II commands are available for use, and they will operate on the file that was most recently selected.

There are times when the contents of a field (such as SUPPLIER in the PARTS file previously mentioned) do not require the facilities of completely dependent records (in independent files), so that dependent fields within the same record would suffice. As an example, suppose you are keeping a database on the latest automobile technology, storing records of press cuttings and references in books that you have seen, relating to things as diverse as injection-moulded plastic bodies and electronically controlled fuel injection. You might design a database format like this:

TOPIC: _____
SYNOPSIS1: _____
SYNOPSIS2: _____
SYNOPSIS3: _____
SOURCE: _____ ISBN: _____
TITLE: _____
DATE: _____
PAGE*NO: _____

If the source of an entry is a magazine, such as *Autocar* or *Off Road Rider*, the ISBN field will be irrelevant, since magazines do not have International Standard Book Numbers. If, however, the source of the entry is a book called *Programming ROM Chips for In-car Computers*, there will be an ISBN. Similarly, if the source is a book or a magazine, there will be a page number. But if the source was the BBC television programme *Wheelbase*, the PAGE NO field would not be needed. Some DBMs, such as Rescue, by Microcomputer Business Systems, allow the presence or absence of any particular field within a record to be either calculated or dependent upon other, previous fields.

Thus, if the SOURCE is not a book, the ISBN field will not be prompted for entry and will not be displayed (either on the screen or in printouts). If the source is neither a book nor a magazine, the PAGE NO field will not be required and will not be displayed. A DBM able to do this can save on memory by eliminating unnecessary fields, and helps to improve the presentation of information. It is still, however, less versatile than a DBM able to relate records in one file to records in another file.

In databases where there may be some root information that remains constant for every record, and some branch information that may or may not be required, we have what is known as a 'two-level hierarchy'. As a general rule, multi-file DBMs can treat two-level hierarchical data as a subset or special case of a multi-file database. For example, a SOURCE reference to a book could relate to a BOOKS file and a SOURCE reference to a programme could relate to a PROGRAMMES file.

There is one final thing to bear in mind. A hand-written card-index file of your own design can be as complex as you care to make it. But there will be serious limitations among the different, commercially available DBMs as to what they can and cannot handle structurally. So before purchasing a DBM for your micro, think hard about what exactly you want it to do and look carefully at the specifications for any product you are considering.

Databases Detailed

NAME	MANUFACTURER	MACHINE	FORMAT	PRICE	COMMENTS
Betabase	Clares	BBC	disk	£25.00	Max record length 2048. Max fields 200. Max file size 99K (40T), 199K (80T)
Database	Gemini Marketing	Spectrum	cassette	£19.95	Card index system. User-defined records.
DFM Database	Dialog	C64	disk	£24.00	15 fields/record. 36-character alphanumeric field. 9-figure numeric field.
Practifile	Computer Software Associates	C64	disk	£44.50	3800 rec/file. Rearranges data for mailing list. Batch entry.
MicroPen	AMSoft	Amstrad	disk	£49.95	Contains built-in word processor and spreadsheet. Max record length 1024. Max no of records in file 32750.

OPTICAL DISC

Perhaps better known by its commercial name, laser disc, an *optical disc* is a storage device on which digital data is etched onto the tracks (spiral or concentric) that can be read by photoelectric cells. The information is stored by etching gaps in the tracks that, when read back, will be interpreted as logical one. Areas that have not been etched will be read as logical zero.

Although at the moment commercial application of optical discs has consisted of providing an alternative storage medium to video cassettes, the potential applications of this technology for home computer users is immense. A standard optical disc can store over two gigabytes (two billion bytes) of information, yet the price of such a device is not much greater than that of a video cassette.

Unfortunately, the drawback with optical discs at present is that the information has to be etched on with a laser and cannot be erased. Thus, at the moment, they are suitable only for pre-purchased programs and not for back-up storage. However, when we consider that many of the newer integrated business packages already require half a megabyte of storage, the idea of buying an optical disc with perhaps only one or two applications packages on it is not so outlandish.

As with many advances in computer technology, the first applications of optical disks were found in arcade games. Arcade players are now able to 'fly' aircraft (computer-generated sprites) through real scenery, which has been filmed (the sprite is superimposed onto the background). The realism of the game is far superior to anything that could be achieved by means of computer-generated graphics.

IAN McKINNELL

Likely Lasers
Optical discs are now finding a wide range of applications in computing as storage mediums. Not only are they being used in their more conventional role of storing TV pictures on which computer graphics can be displayed, but they are also being used to store computer programs which can be loaded and run. The drawback to the technology at present is that optical discs are confined to 'read only' memory.

OPTIMISATION

This refers to the process by which a program is written in order to produce the best solution to a problem. But the best solution may not necessarily be the same in every case. For example, some programs may require maximum speed whereas others may need to make the best possible use of available memory space. A number of techniques have evolved to achieve the various aims of *optimisation.* It is not possible to give explicit examples, since the optimisation of a program will vary with the restrictions placed on it by different situations. However, it is possible to illustrate some general principles.

Optimisation generally consists of removing or streamlining program processes that are unnecessary. But good programming practice should produce the optimum result anyway. The assignment of variables is a good example. Once a variable has been assigned, unless it is altered, there is no need to reassign the same value to it. Thus, the initialisation of variables in a program should not take place within loops or commonly used subroutines.

OR

This is one of the six Boolean logical functions. The result of an *OR* operation is true if either of the inputs are true. The OR function is more properly known as an 'Inclusive–OR' to distinguish it from the Exclusive–OR (XOR) function, the latter precluding the possibility of both inputs being true and producing a true result.

ORDERED PAIR

An *ordered pair* is two numbers that are always given in a set order. For example, when using Cartesian geometry, we refer to the co-ordinates (x,y) always in the same order, with x representing the horizontal axis and y the vertical. The advantage of this is that when we assign a value to the first variable in an ordered pair, we can assume that everyone understands which axis we are defining. Although ordered pairs are the most commonly used, it is also possible to refer to ordered triplets or, indeed, any other grouping.

OSCILLOSCOPE

This is a device that provides a visual representation of a wave pattern. An *oscilloscope* receives an input in the form of an electrical signal and displays the wave form on a cathode-ray tube (CRT) although some of the newer oscilloscopes use LCD technology. Unlike a conventional television screen, an oscilloscope CRT does not build up the picture by the method of raster scanning the horizontal lines. The electron beam is instead deflected in a vertical direction as it moves horizontally across the screen, thus creating the displacement of the amplitude as it varies with time. Oscilloscopes have a wide range ot test applications, from examining the properties of sound waves to calibrating and testing electronic timing devices.

SIX-PACK ROBOT

Currently available robotics kits, though fun to build and interesting to play with, provide no lasting stimuli for the home hobbyist. Fischertechnik has eliminated this problem by introducing a kit that allows you to build either the six designs provided or else any of your own – surely a check against losing interest in these fascinating devices.

Although many people within the home computer industry have been predicting for some time that the next big growth area within the industry will be home robotics, the predictions have still yet to come true. The reasons for this are not hard to find. Despite the fact that over the last year or so numerous 'home robots' have appeared on the market, they have suffered from a few major problems that have prevented the public from being convinced that an introduction to robotics is an interesting and worthwhile endeavour.

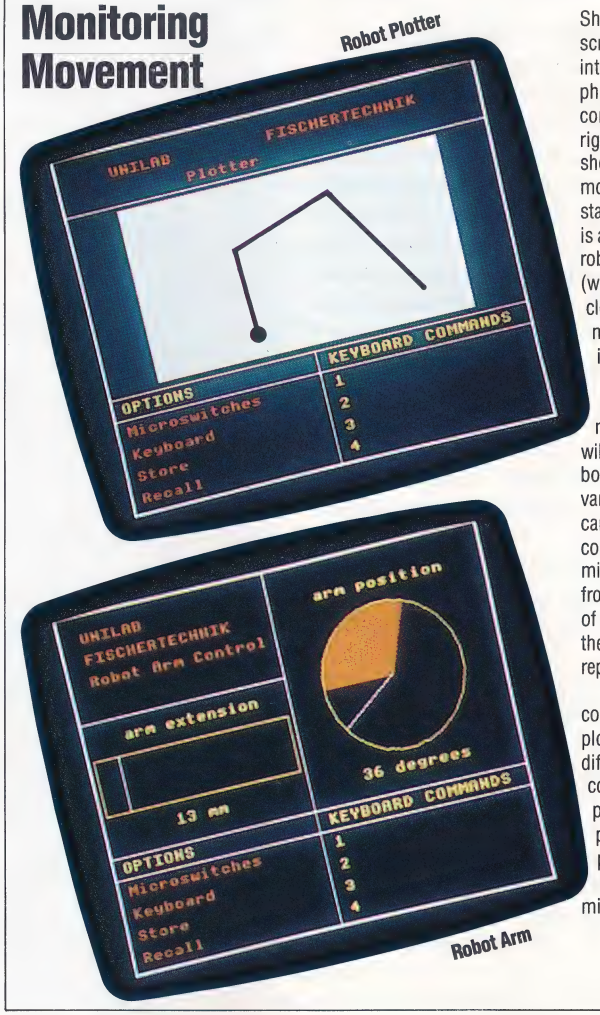
On the one hand, there are the easily built robots such as the Movits (see page 1084), which can be assembled in a few hours by somebody without any knowledge of robotics or even electronics. The problem here is that these devices are strictly limited in their applications and the user will probably soon lose interest in the machines. On the other hand, more expensive robots are available that generally require extensive knowledge of robotics and electronics in order to operate them. Even then, many of these more advanced robots have only a single purpose. Once again, this means that when all the avenues have been explored, the robot is left to collect dust on the shelf.

Thus it seems obvious that what is required is a robot that can be easily constructed which has a number of different functions that can each be explored by the user. This is the claim that is made for the Fischertechnik Robotics Kit. The idea behind the kits is fairly simple. For many years now there have been a number of electronics kits, intended for children, enabling them to construct simple circuits to form, say, a basic radio. This same kit can then be taken apart and reconstructed to form a simple electronic timing device or burglar alarm. Simultaneously, generations of children have been brought up on Lego kits that can be used and reused to build an enormous number of different models from the same plastic building blocks. In designing its robotics kits, Fischertechnik has adopted both of these ideas and merged them with modern computer technology to develop a product that could well become the popular breakthrough for robotics that so many have been predicting.

The Robotics kit allows the user to build a number of different robot devices that can be run from a home micro. These robots range from a robot arm and a machine that sorts items of varying lengths into order, to a plotter and a graphics input device. Once the user has experimented with one particular robot, it can be taken apart and rebuilt to form another. In order to run the robots from a computer, an interface is required to turn the computer's digital signals into those which can be used to operate the kit's electric motors.

Consisting of a number of plastic pieces, the construction kit components can be slotted together — rather like a Lego kit — to form various models. These pieces take the form of blocks that can be linked together to form the longer pieces of the arms or used to hold the control units. Other pieces for building the mechanical units are

Monitoring Movement



Shown here are the opening screen displays of the Unilab interface software. The top photograph is of the robot arm control program. In the top right-hand corner is a diagram showing the horizontal movement of the arm. The starting position of the robot is assumed to be zero. As the robot arm moves to the right (with the line moving clockwise on the screen) the number of degrees will increase. Similarly, when the arm extension is moved the line within the bar and the number of millimetres moved will change accordingly. At the bottom of the screen are the various control options that can be applied. These are controlled from the microswitches on the board or from the keyboard. Sequences of movements can be stored in the computer and recalled and replayed later on.

The same system of commands is used with the plotter robot, the only difference being that extra commands are added to the plotter to raise and lower the pen onto the paper. As the plotter moves around the paper its movements will be mirrored on the screen

cogged wheels and screw blocks to turn the cogs. There is also a large plastic board measuring 260 by 187mm, which is used either as a base for the robot itself or, in the event of the plotter or graphics tablet being incorporated, as the base for the paper. The electrical control and power system also need assembling from separate components. The kits include a number of male and female sockets, eight switching devices and a pair of electric motors. Also provided is about a metre of 20-way ribbon cable and a small extra piece of cable for constructing 'patches'.

Overall, the pieces are sturdily constructed and look as though they could last for many years of service. One exception to this appears to be the potentiometers. The base plate of one of them broke on construction, and although this did not affect the running of the arm (shown in the photograph) because the baseplate could be pressed back into position, it does raise a question over their long-term reliability. Another point concerning the potentiometers is that unlike the rest of the wiring, which can be screwed into sockets, there is no method of attaching the bare wires to them other than by wrapping them around the connectors. In this case it is advisable for the hobbyist intending to make a lot of use of the kit to solder the joints permanently to prevent short circuits that could damage the potentiometers.

FITTING THE COMPONENTS

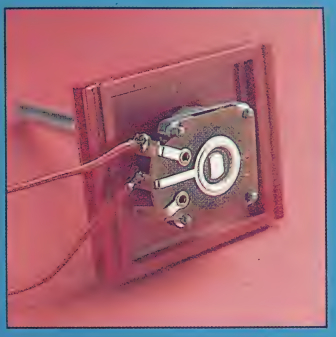
Construction of the robots is quite simple and the pieces for the most part are chunky enough for the most unskilled fingers to manage. The major problem in construction is due to the manual, which is probably the greatest weakness of the entire system. Although it does have its good points — for example, the easy-to-follow circuit connections — the manual relies heavily on rather low-quality photographs of the partly constructed models. Photographs of the pieces required for a particular section are displayed along with partly exploded views of the robot under construction. The trouble is that these photographs are not particularly detailed and lack any annotations, meaning that builders will often find themselves peering at a certain photograph trying to find where a particular piece is supposed to go. Furthermore, as the photos show the robot from only a single direction, you will often have to guess where a piece is fitted when its position is obscured from view.

What is perhaps worse is the fact that the photos are in black and white. This means that when you eventually get to the final part, which is fitting the circuit wiring on the robot, it is next to impossible to figure out the correct wiring procedure. This is particularly strange if you consider that the circuit depends largly on the colour coding of the ribbon cable. The minimal annotations provided with the wiring diagrams are not much help either, merely stating that, for example, wires E3 to E8 belong in a particular section.

That said, it should be pointed out that the particular robot shown was built by someone with no knowledge of robotics or electronics who did manage to get the machine working, although it would have been nice if Fischertechnik did not rely on people's common sense and simply provided a better construction manual.

Once the robot has been completed, the next task is to connect it to the interface. Here, happily, the procedure is much simpler. The interface board consists of a number of sockets divided into sections to hold the leads from the ribbon cable. The interface shown in the photograph is that for the BBC Micro, although interface boards for other computers will probably not be radically different. On the right-hand corner of the board are the switch inputs. The logic of the switch inputs is governed by the data register and data direction

Electromagnet
The metal bar on top of the electromagnet slides into the end effector of a robot arm. The magnet can be switched on and off, enabling the robot to lift the small metal plates provided with the robotics kit



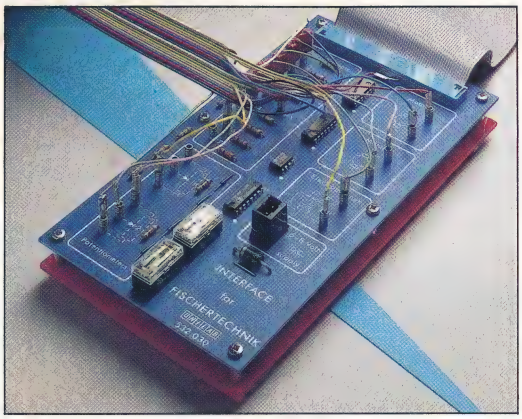
Potentiometer
The potentiometers are used to provide feedback to the computer so that the correct position of the robot can be ascertained

Motor
The motor is entirely contained within a plastic casing. The connection to the motor is via two sockets on the casing. The rotating screw is used to drive cog mechanisms that drive the robots

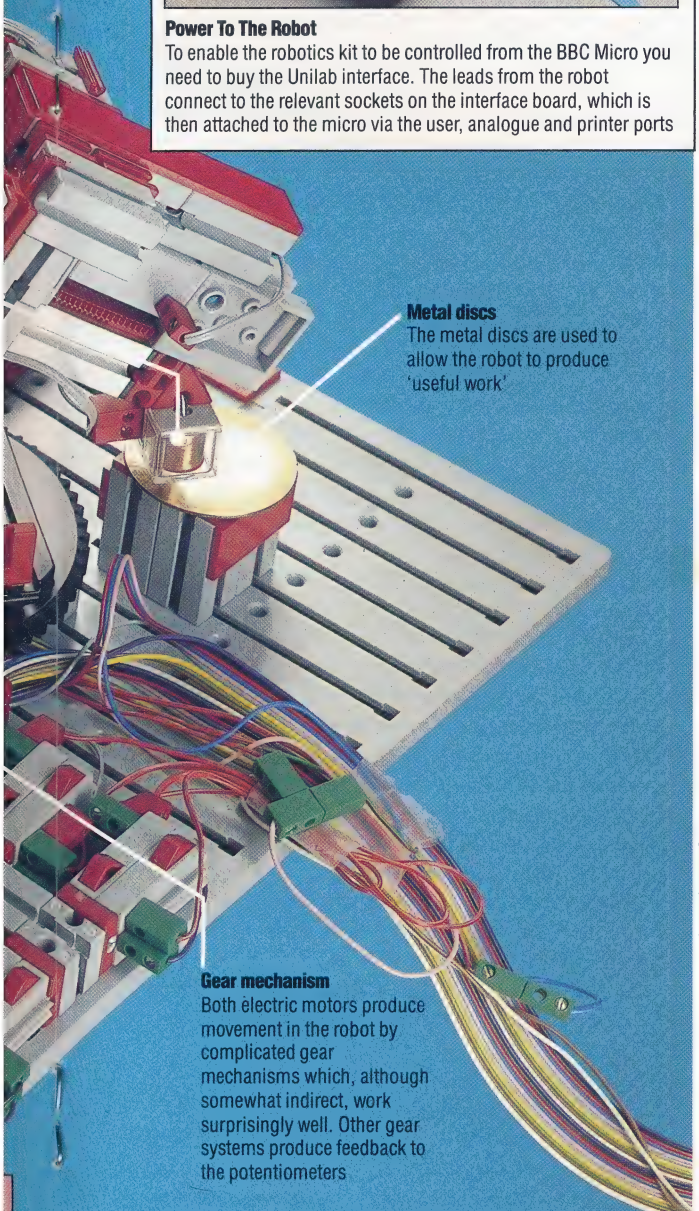
Lights
Whenever one of the three main control systems (magnet, horizontal or vertical motors) is engaged one of these bulbs will light up to inform the user which control is being used

IAN McKINNELL

Switch
The eight switches correspond to the eight data lines on the BBC's user port. By pressing a combination of these switches a number will be generated on the user port register, which by controlling the data direction register can be used to control the robot directly



Power To The Robot
To enable the robotics kit to be controlled from the BBC Micro you need to buy the Unilab interface. The leads from the robot connect to the relevant sockets on the interface board, which is then attached to the micro via the user, analogue and printer ports



Metal discs
The metal discs are used to allow the robot to produce 'useful work'

Gear mechanism
Both electric motors produce movement in the robot by complicated gear mechanisms which, although somewhat indirect, work surprisingly well. Other gear systems produce feedback to the potentiometers

CHRIS STEVENS

register of the BBC Micro and is similar in operation to the buffer box project designed in our Workshop series. Below that is a pair of potentiometers, which allows the computer to monitor the power being fed to the switches and light bulbs. Control of the potentiometers is measured through the analogue port.

On the other side of the interface board are connections to control the motors and the electromagnet (the latter is used by the robot arm to pick up metal objects). Finally, there is 6 to 8v power supply socket that powers the entire system.

The interface is connected to the BBC Micro via three separate ports — the user port, the analogue port and the printer port — used to control the motors and electromagnet. Because the system is designed for inexperienced users, the interface board has been constructed to guard against the possibility of incorrect wiring that could lead to voltages being applied to the computer's interfaces which may cause damage to the computer.

The manual that is provided for the interface (written by the product's manufacturer, Unilab) is much superior to the Fischertechnik Robotics manual. There is a detailed explanation on how to set up the interface and how to run the software provided with the system. There is also a brief summary of the control addresses used to control the system, thus allowing the robots to be controlled from BASIC. However, the wiring diagram appears to be at odds with the Fischertechnik circuit. Although an accompanying note concerning different ribbon cables is provided with the interface, it does not appear to clear up all the problems and once again you will be left to your wits on how to wire the circuit correctly.

FISCHERTECHNIK ROBOTICS KIT
PRICE
Robotics Kit £69.95, Interface (sold separately) £49.95 (both inc VAT)
MOVEMENT
The robots are powered by one or two DC electric motors
CONTROL
Either from the computer keyboard or via the eight switches on board the robot itself
POWER SOURCE
A single 6v PJ996/4R25 battery (not included)
SUPPLIERS
Commotion, Computer Operated Motion, 241 Green Street, Enfield EN3 7SJ, Middlesex

THE PROF LANGUAGE

Provided with the interface is a cassette containing three programs to assist in the control of the robots. The first two are specialised programs that control the arm and the plotter, allowing the external devices to be controlled either from the keyboard or from the control switches on the robot itself. The third program is Prof, a generalised operating system through which you can directly command the robots. It is essentially an extension of BASIC with a few extra commands added. Many of the extra words introduced by Prof are variable names like Motor and Magnet for the control address, whereas others are introduced to assist the editing of sequences of actions to be performed by the robots.

Although Prof is fairly comprehensive and allows conditional loops and structured programming, it is confined mostly to output control. Thus the graphics panel, an input device, is unsupported by Prof because the language makes no provision for storing information from input devices. This necessitates your having to create your own programs from BASIC.

Despite the problems with the documentation, the Robotics kit and interface collectively is a very worthwhile system. The robots work surprisingly well and a working model can be built from scratch in just a few hours. And, the user is not just restricted to the six designs provided. Once familiarity has developed with the system, there are any number of projects that could be constructed. At over £100 it is certainly not a pocket-money toy, and once the bugs in the documentation have been removed, the Fischertechnik Robotics kit and interface will be an ideal introduction for anyone who is generally interested in looking into the field of robotics.

PLAGUED WITH PROBLEMS

We have seen how the 13 possible minor events can affect the length and ease of the trading voyage to the New World. In this instalment, we begin the implementation of six possible major events that will have far more influence, such as encountering and rescuing the crew of a drifting lifeboat or dealing with an attack of the plague.

In this instalment, we begin the programming for the major contingencies. There will be six of these events, each of which may occur once during the voyage. They are termed 'major' because their effects on the voyage are more complex and far-reaching than those of the minor events. One of the six events will be selected randomly at the start of each week, but if the selected event has already occurred during the voyage, there will be no major event during that week.

Line 49 DIMensions an array, M(), which will be used to flag whether each of the six major contingencies has occurred. The program is sent to subroutine 6500, which generates an event from the main program loop by a GOSUB at line 870. Line 6508 selects a random number between 1 and 10 and, because there are only six possible events, there is a 60 per cent chance that an event will be selected: if 7, 8, 9 or 10 is generated, no major contingency will occur. The odds can be altered if desired by adjusting the range of the random factor on this line.

For example, changing line 6508 to:

```
X=INT(RND(1)*8)+1
```

increases the chance of selecting a major event to 6 in 8, or 75 per cent. It should be realised, however, that if a major event is selected and the corresponding flag in the array M() set, then the probability of selecting a second major event reduces to 5 in 8, or 62.5 per cent. The probability of selecting a third major event diminishes further still to 4 in 8, or 50 per cent. Thus the more events that occur, the less is the probability of another, so far unselected, event occurring.

The event subroutine is selected using the ON X GOSUB statement. This is used in the same way as the ON X GOTO statement selected the minor contingencies. If X, the randomly generated number, is one, it will send the program to the first line number in the list of subroutines following the statement, subroutine 6530 (the sighting of a lifeboat). If X is 2, it will go to the second address in the list, subroutine 6700 (the deadly disease). We are covering only the first two contingencies in this instalment but in future sections we will need to add the addresses of the remaining four subroutines to line 6510.

In the first subroutine, a lifeboat is sighted through the telescope. The boat contains four people and a large chest, and the player must decide whether or not to take them on board. The diversion will add two days to the journey and there will be four extra mouths to feed, but perhaps the extra manpower will be welcome if the crew is understrength. The program first checks if the lifeboat has been sighted previously by looking at the first element in array M() in line 6535. If it has been set to 1, the event has already occurred, and it returns to the main program. If it is 0, line 6540 sets M(1) to 1 to prevent the event recurring later in the game. As mentioned earlier, this check is used at the start of each major contingency.

The player is asked whether the shipwreck crew should be picked up at line 6576. Line 6580 checks if the first character of the reply is Y or N, and, if not, returns to 6568 to ask the question again. If the reply is No, the lifeboat floats away and disappears and control returns to the main program. If compassion wins the day and the player does decide to save the stranded crew, the program checks to see whether there is enough room by first ascertaining if there is already a full complement of crew members aboard at line 6625. If so, there will be no room on the ship (indicated by CN=16), and control returns to the main program. To see if there is room for any of the survivors, line 6630 counts the empty spaces on the ship by setting the value of X to 16 minus the crew number, CN, and the program decides if there is room for all four survivors at line 6630 by checking if X is greater than 3. If so, they are picked up, but if X is 3 or less, the player is told there is room for only X more people, who are then brought aboard.

RATINGS ON BOARD

The survivors from the lifeboat are then incorporated into the crew strength/type array in the loop from line 6638 to 6697. The value of X is set initially to 0 and used to keep a count of extra crew members. It checks through the array TS(,) to see if the crew type is 0, signifying a place in the array for an additional crew member. If a place is found, the value of X is incremented by 1, indicating that one of the survivors has been added. When X reaches 4, all the survivors will have been incorporated into the crew array and so line 6655 sets the value of T to 16, causing the program to drop out of the loop when it reaches NEXT. If there is space for less than four extra crew, the program will drop out of the loop when the

IAN McKINNELL

array has been checked 16 times. The extra members are also added to the value of variable CN,the crew count. Each new member is assigned a random value for crew type and strength within the strength/type array TS(,); type is selected from the range 1 to 5 and strength from the range 50 to 99. This means that the extra crew will be rated as healthy or very healthy.

There was also a chest in the lifeboat containing some provisions. As the crew type designations for the four survivors from the lifeboat are selected randomly, this means that, say, a previously doctor-less ship might be able to enlist the services of a doctor from among the lifeboat crew. A loop is set up at line 6685 for each of the four provision types, and line 6690 generates a random number between 10 and 19, representing the quantity of each provision. The quantity, units (held in U$(), representing kilos or barrels) and type of provision are printed at line 6692. If a provision has been exhausted during the current week line 6693 resets the quantity from -999 to 0 to enable the addition to be recorded and line 6694 adds the new provisions to the present stock.

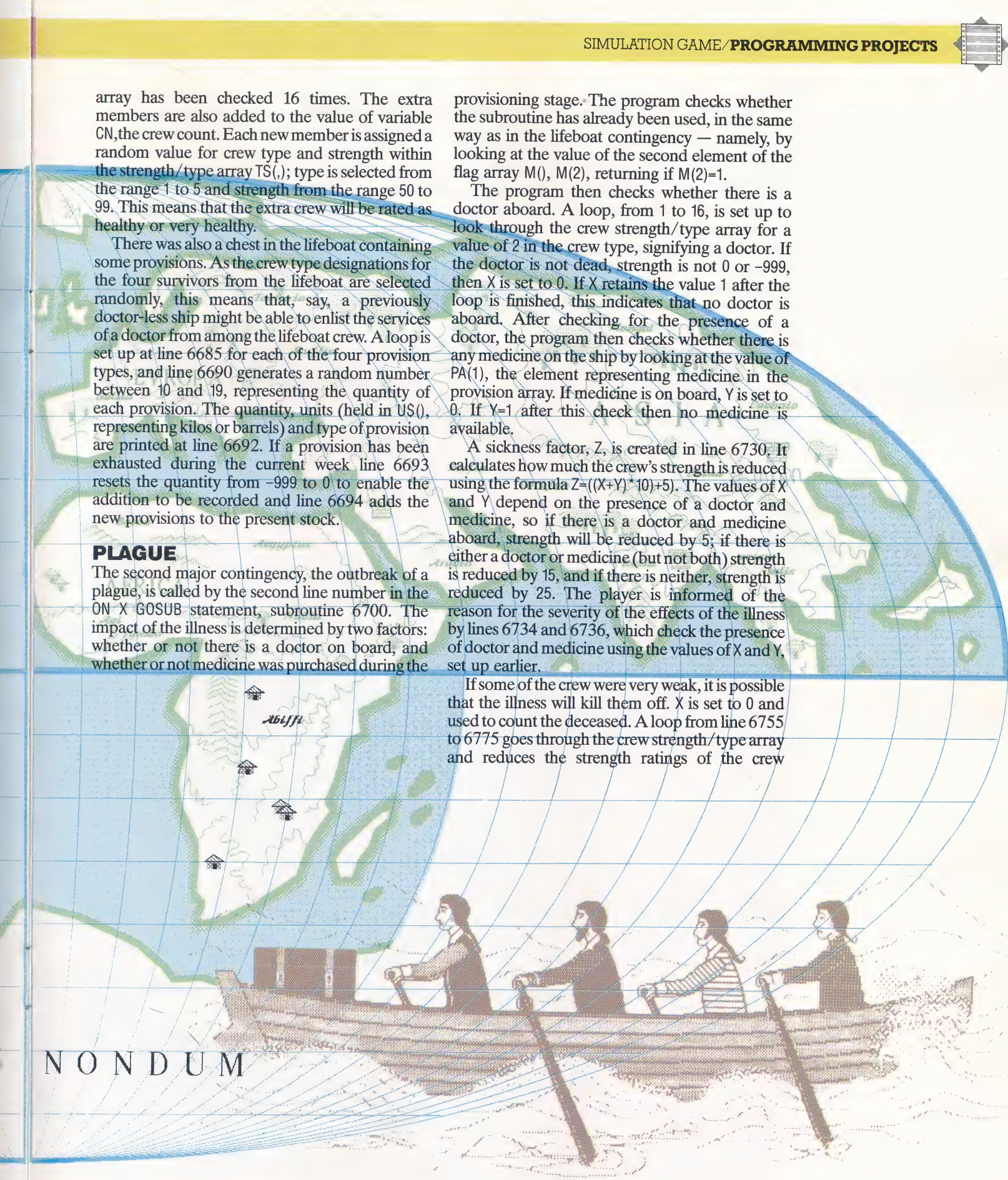
PLAGUE

The second major contingency, the outbreak of a plague, is called by the second line number in the ON X GOSUB statement, subroutine 6700. The impact of the illness is determined by two factors: whether or not there is a doctor on board, and whether or not medicine was purchased during the provisioning stage. The program checks whether the subroutine has already been used, in the same way as in the lifeboat contingency — namely, by looking at the value of the second element of the flag array M(), M(2), returning if M(2)=1.

The program then checks whether there is a doctor aboard. A loop, from 1 to 16, is set up to look through the crew strength/type array for a value of 2 in the crew type, signifying a doctor. If the doctor is not dead, strength is not 0 or -999, then X is set to 0. If X retains the value 1 after the loop is finished, this indicates that no doctor is aboard. After checking for the presence of a doctor, the program then checks whether there is any medicine on the ship by looking at the value of PA(1), the element representing medicine in the provision array. If medicine is on board, Y is set to 0. If Y=1 after this check then no medicine is available.

A sickness factor, Z, is created in line 6730. It calculates how much the crew's strength is reduced using the formula Z=((X+Y)*10)+5). The values of X and Y depend on the presence of a doctor and medicine, so if there is a doctor and medicine aboard, strength will be reduced by 5; if there is either a doctor or medicine (but not both) strength is reduced by 15, and if there is neither, strength is reduced by 25. The player is informed of the reason for the severity of the effects of the illness by lines 6734 and 6736, which check the presence of doctor and medicine using the values of X and Y, set up earlier.

If some of the crew were very weak, it is possible that the illness will kill them off. X is set to 0 and used to count the deceased. A loop from line 6755 to 6775 goes through the crew strength/type array and reduces the strength ratings of the crew



affected by the disease. However, the illness does not affect all the crew members. Line 6756 generates a random number which if less than 0.3 means that the strength of the crew member in the current element of the array is not affected by the disease. This gives each of the crew about a 30 per cent chance of escaping illness (you may wish to alter this factor, or possibly set it up randomly). The next line checks whether the person is already dead, and hence beyond the effects of the disease. If the crew member is alive, strength is reduced by Z units and a check is made to see whether the illness killed the crew member by reducing the strength rating below 0. If so, it is reset to 999 where it will be picked up in the end of week report and the value of X, the number killed by the disease, will be increased by 1. If there was medicine available, indicated by Y=0, line 6776 reduces the quantity by half and rounds it up to the nearest whole bottle. If none of the crew has been killed in the epidemic (X=0), then control is returned to the main program. Otherwise, the player is informed how many crew were fatally affected.

In the next module, we consider more of the major events that can occur during the voyage, including a problem with the ship's rudder.

Module Eight: Two Major Events

Dimension Flag Array

```
49 DIMM(6):REM INDS TO SHOW IF MAJOR EVENTS HAVE B
EEN DONE
```

Addition To Main Voyage Loop

```
870 GOSUB6500:REM GOTO MAJOR CONTINGENCY
```

Major Contingency Selection Routine

```
6500 REM MAJOR CONTINGENCIES
6508 X=INT(RND(1)*10)+1
6510 ON X GOSUB6530,6700
6520 RETURN
```

Major Contingency Routines

```
6530 REM LIFEBOAT
6535 IFM(1)=1THENRETURN
6536 PRINTCHR$(147)
6540 M(1)=1
6550 S$="A LIFEBOAT IS SIGHTED*":GOSUB9100
6552 S$="DRIFTING IN THE DISTANCE*":GOSUB9100
6554 PRINT:GOSUB9200
6556 S$="THROUGH THE TELESCOPE YOU SEE*":GOSUB9100
6558 S$="THAT IT CONTAINS:*":GOSUB9100
6560 PRINT:GOSUB9200
6562 S$="4 PEOPLE*":GOSUB9100
6563 GOSUB9200
6564 S$="AND A LARGE CHEST!*":GOSUB9100
6566 PRINT:GOSUB9200
6568 S$="IF YOU CHANGE COURSE*":GOSUB9100
6570 S$="TO PICK THEM UP*":GOSUB9100
6572 S$="IT WILL TAKE YOU TWO EXTRA DAYS*":GOSUB91
00
6574 PRINT:GOSUB9200
6576 S$="DO YOU WANT TO PICK THEM UP (Y/N)*":GOSUB
9100
6578 INPUTI$:I$=LEFT$(I$,1)
6580 IFI$<>"Y"ANDI$<>"N"THEN6578
6585 IFI$="Y"THEN6600
6588 PRINT:GOSUB9200
6590 S$="THE LIFEBOAT DISAPPEARS....*":GOSUB9100
6592 PRINT:GOSUB9200
6594 S$=K$:GOSUB9100
6596 GETI$:IFI$=""THEN6596
6599 RETURN
6600 PRINT:GOSUB9200
6610 EW=EW+2/7
6625 IFCN<>16 THEN6630
6627 S$="YOU CAN'T PICK THEM UP*":GOSUB9100
6628 S$="YOU HAVE NO ROOM ON THE SHIP*":GOSUB9100
6629 GOTO6592
6630 X=16-CN:IFX>3THEN6635
6632 S$="YOU ONLY HAVE ROOM FOR*":GOSUB9100
6633 PRINTX;"MORE PEOPLE ON THE SHIP"
6634 PRINT:GOSUB9200
6635 S$="YOU PICK UP:*":GOSUB9100
6638 X=0
6640 FORT=1TO16
6645 IFTS(T,1)<>0THEN6679
6650 X=X+1
6655 IFX>4THENT=16:GOTO6679
6660 CN=CN+1
6665 TS(T,1)=INT(RND(1)*5)+1
6668 TS(T,2)=INT(RND(1)*50)+50
6670 PRINT"1 ";C$(TS(T,1))
6679 NEXT
6680 PRINT:GOSUB9200
6682 S$="THE CHEST CONTAINS:*":GOSUB9100
6685 FORT=1TO4
6690 X=INT(RND(1)*10)+10
6692 PRINTX;U$(T);"S OF ";P$(T)
6693 IFPA(T)=-999THENPA(T)=0
6694 PA(T)=PA(T)+X
6695 NEXT
6699 GOTO6592
6700 REM PLAGUE STRIKES
6705 IFM(2)=1THENRETURN
6706 PRINTCHR$(147)
6710 M(2)=1
6712 S$="A PLAGUE STRIKES*":GOSUB9100
6714 PRINT:GOSUB9200
6716 X=1
6718 FORT=1TO16
6720 IFTS(T,1)=2ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=0:T=16
6722 NEXT
6724 Y=1
6726 IFOA(1)<>0ANDOA(1)<>-999THENY=0
6730 Z=((X+Y)*10)+5
6732 I$="YOU HAVE ":IFX=0ANDY=0THEN6740
6734 IFX=1THENS$="WITH NO DOCTOR*":GOSUB9100:I$="A
ND "
6736 IFY=1THEN S$=I$+"NO MEDICINE*":GOSUB9100
6740 S$="MANY OF THE CREW ARE AFFECTED*":GOSUB9100
6745 PRINT:GOSUB9200
6750 X=0
6755 FORT=1TO16
6756 IFRND(1)<.3THEN6775
6760 IFTS(T,2)=0ORTS(T,2)=-999THEN6775
6765 TS(T,2)=TS(T,2)-Z
6770 IFTS(T,2)<1THENTS(T,2)=-999:X=X+1
6775 NEXT
6776 IFY=1THEN6780
6777 S$="1/2 YOUR MEDICINE IS USED*":GOSUB9100
6778 OA(1)=INT((OA(1)/2)+.5)
6780 PRINT:GOSUB9200
6785 IFX=0THEN6797
6790 PRINT"AND";X;
6792 S$="CREW MEMBERS DIE*"
6794 IFX=1THENS$="CREW MEMBER DIES*"
6795 GOSUB9100
6796 PRINT:GOSUB9200
6797 S$=K$:GOSUB9100
6798 GETI$:IFI$=""THEN6798
6799 RETURN
```

Basic Flavours

Spectrum:
Make the following changes:

```
6510 IF X = 1 THEN GO SUB 6530
6511 IF X = 2 THEN GO SUB 6700
6536 CLS
6578 INPUT I$ : LET I$ = I$ (1 TO 1)
6596 LET I$ = INKEY$ : IF I$ = "" THEN GO TO 6596
6706 CLS
6798 LET I$ = INKEY$ : IF I$ = "" THEN GO TO 6798
```

BBC Micro:
Make the following changes:

```
6536 CLS
6596 I$ = GET$
6706 CLS
6798 I$ = GET$
```

ELBOW GREASE

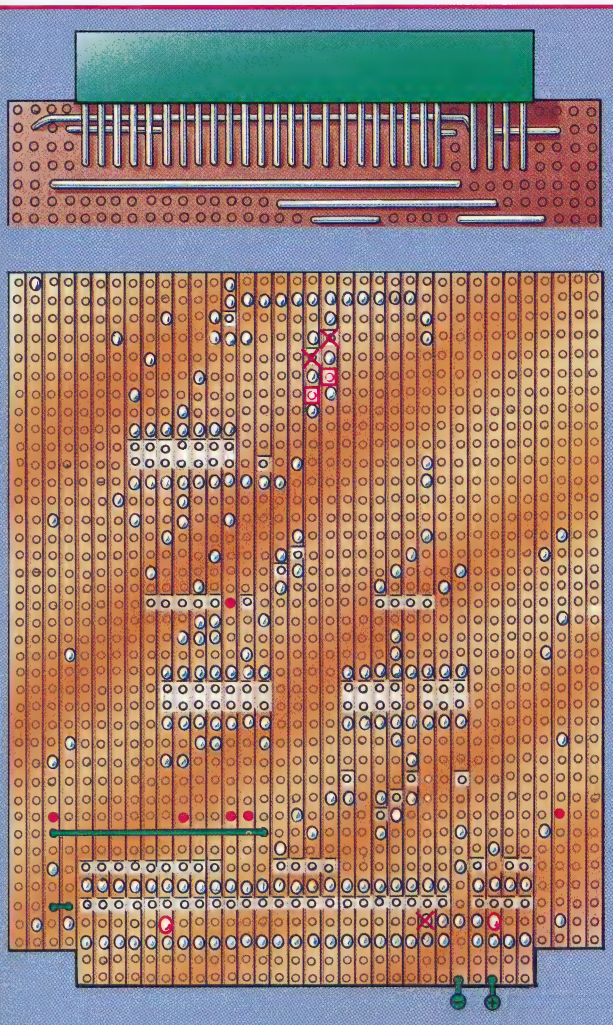
The differences in the architecture, operating system and BASIC dialect between the BBC Micro and Commodore 64 mean that creating programs to control the Workshop robot arm will be different in many respects. We now look at the Commodore.

The Commodore operating system generates regular interrupts to perform housekeeping operations, such as scanning the keyboard. These interrupts occur regularly at one sixtieth of a second intervals. The Commodore machine code program intercepts the interrupt vector to perform its own routine first before passing control to the normal interrupt routine. Such a program is called a 'wedge' (it is dealt with in more detail on page 1139). The first section of the machine code is therefore concerned with swapping the normal IRQ vector with the vector for our routine.

Once our vector has replaced the normal IRQ vector, our event handler will be implemented each time an IRQ interrupt occurs. (The event handling code itself is largely the same as the multiple-servo control program given on page 923). Essentially, this code sends pulses of current down the data lines from the user port to the motors. The duration of each pulse informs the corresponding motor what angle to adopt. The machine code routine receives the information that determines the length of the motor pulses from an eight-byte table, called ANGLE. As with the BBC Micro version, an extra piece of code is used to load the ANGLE locations smoothly from a second table, NEWPOS, which is the table that is loaded from the BASIC controlling program.

The arm sequence programmer uses the machine code program just described to drive the robot arm. Similar to the BBC Micro version, this program allows you to control the arm from the keyboard, saving key positions that can be replayed or stored on disk or tape. There are, however, one or two important differences between this and the BBC Micro version. First, when controlling the robot arm, the motors have a tendency to oscillate through a given position. This is because the Commodore's VIC-II video chip interferes with the regular generation of interrupts. The simplest way to avoid this problem is to blank out the screen when using the wedge program. The subroutines at lines 7000 and 8000 clear the screen before entering the wedge and restore it when the wedge is exited.

The second problem is that Commodore BASIC makes checking a large number of keys (and branching accordingly) a slow process. The final machine code subroutine is therefore used to check the ASCII code of a keypress against a table of possible keys that have a function in the table. When a match is found in the table, it is passed as a number that can be used by BASIC's ON . . . GOSUB to call the correct routine.



KEVIN JONES

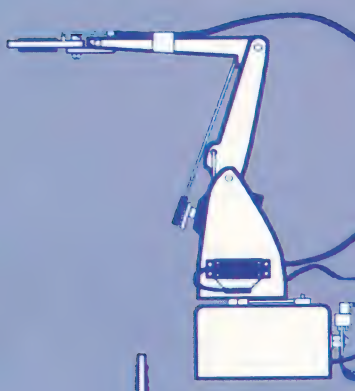
Erratum

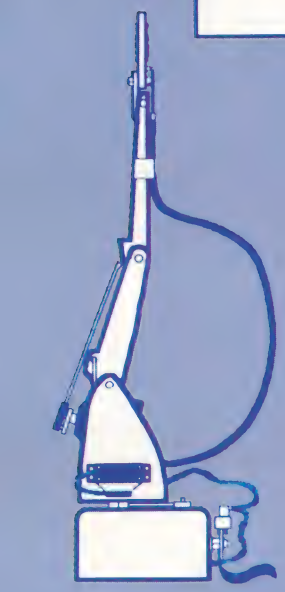
The board layout diagram for the Spectrum Interface on page 1013 is incorrect. The following changes are shown: The long flying lead should be one hole further to the right, at its right-hand end, as should the left-hand end of the adjacent wire link. The single bridging link on the second row should be moved one place to the right. The right-hand end of the short wire link on the left of row 2 should be located two holes further to the right. Note also the corresponding changes to the copper side of the board

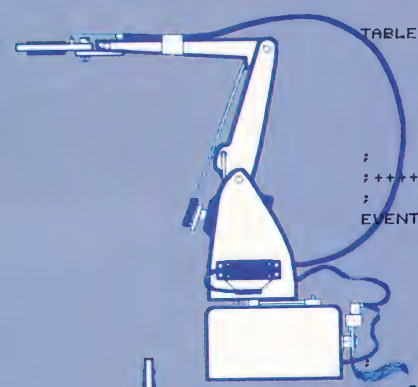
Commodore 64 Arm Sequence Programmer

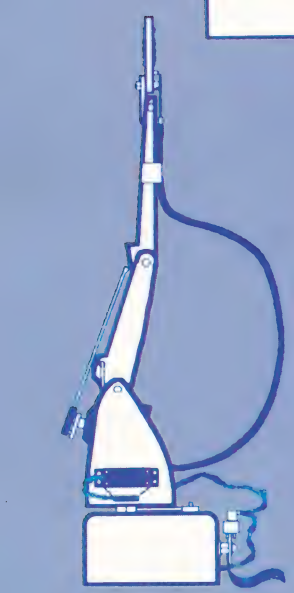
BASIC Loader

```
10 REM ************************
20 REM **                    **
30 REM **  BASIC LOADER FOR  **
40 REM ** CBM ARM CONTROLLER **
50 REM **                    **
60 REM ************************
65 :
70 FOR I=49155 TO 49394
75 READ A:POKE I,A:CC=CC+A
80 NEXT I
90 READ CS:IF CS<>CC THEN PRINT"CHECKSUM ERROR":STOP
95 :
100 DATA169,255,141,3,221,169,62,141,1
110 DATA192,169,192,141,2,192,120,173
120 DATA20,3,174,1,192,141,1,192,142
130 DATA20,3,173,21,3,174,2,192,141,2
140 DATA192,142,21,3,169,16,133,251
150 DATA169,193,133,252,169,255,160,0
160 DATA145,251,136,208,251,88,96,8,72
170 DATA152,72,138,72,169,255,141,1
180 DATA221,162,7,169,255,24,106,72
190 DATA188,0,193,49,251,145,251,104
200 DATA202,16,243,160,48,136,208,253
210 DATA169,255,160,0,49,251,141,1,221
220 DATA200,208,248,162,7,169,255,188
230 DATA0,193,145,251,202,16,248,104
240 DATA170,104,168,104,40,108,1,192
250 DATA162,0,160,0,189,8,193,221,0
260 DATA193,240,14,176,6,222,0,193,76
270 DATA156,192,254,0,193,76,156,192
280 DATA200,232,224,4,208,228,32,169
290 DATA192,192,4,208,217,96,138,72
300 DATA152,72,160,255,174,0,192,136
310 DATA234,234,208,251,202,208,248
320 DATA104,168,104,170,96,0,255,255,0
330 DATA0,255,255,0,0,255,255,0,0,255
340 DATA255,0,0,72,138,72,152,72,32
350 DATA228,255,201,0,240,249,162,0
360 DATA221,191,192,240,7,232,224,17
370 DATA208,246,162,255,142,207,192
380 DATA104,168,104,170,104,96
390 DATA33097:REM*CHECKSUM*
```









Assembly Listing

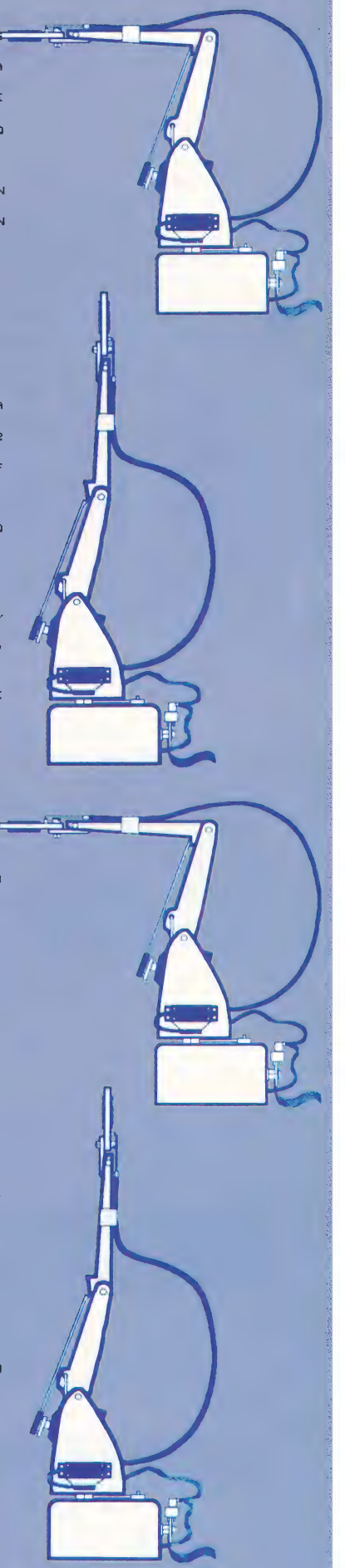
```
;++++++++++++++++++++++++
;++++++++++++++++++++++++
;++                    ++
;++ CBM ARM CONTROLLER ++
;++                    ++
;++++++++++++++++++++++++
;++++++++++++++++++++++++
;
PORT = 56577 ;USER PORT DATA REGISTER
DDR=56579    ;USER PORT DATA DIRN REG
ANGLE=$C100  ;ANGLE VALUE LOCATION
NEWPOS=$C108 ;POSITION TABLE
MOTTAB=$C110 ;MOTOR LOOK UP TABLE
ZPAGE=$FB    ;0 PAGE POINTER TO TABLE
IRQVEC=$0314 ;IRQ VECTOR LOBYTE
;
*=$C000
DELFAC *=*+1 ;STORAGE FOR DELAY FACTOR
OURVEC *=*+2 ;STORAGE FOR OUR VECTOR
      LDA #$FF
      STA DDR      ;SET DDR TO OUTPUT
      LDA #<EVENT
      STA OURVEC   ;POINT TO EVENT
      LDA #>EVENT  ;HANDLER
      STA OURVEC+1
;
      SEI          ;INTERRUPTS OFF
      LDA IRQVEC   ;SWAP EXISTING
      LDX OURVEC   ;IRQ VECTOR FOR
      STA OURVEC   ;OUR VECTOR
      STX IRQVEC
      LDA IRQVEC+1
      LDX OURVEC+1
      STA OURVEC+1
      STX IRQVEC+1
;
;++++ INITIALISE TABLE ++++
;
      LDA #<MOTTAB
      STA ZPAGE
      LDA #>MOTTAB
      STA ZPAGE+1
;
      LDA #$FF
      LDY #$00
TABLE
      STA (ZPAGE),Y
      DEY
      BNE TABLE
      CLI        ;INTERRUPTS ON
      RTS
;
;++++ EVENT HANDLER ++++
;
EVENT
      PHP
      PHA        ;SAVE REGISTERS
      TYA        ;ON STACK
      PHA
      TXA
      PHA
;
;++ START PULSE, FOR SOME MOTORS IT
;MAY BE POSSIBLE TO START BEFORE FILLING
;TABLE AND SO REDUCE WAIT LOOP BELOW ++
;
      LDA #$FF
      STA PORT
;++ FILL TABLE WITH EXCEPTIONS ++
      LDX #$07
      LDA #$FF
      CLC
EXCEPT
      ROR A
      PHA        ;BIT PATTERN
      LDY ANGLE,X ;GET MOTOR X OFFSET
      AND (ZPAGE),Y ;KEEP EXISTING PATTERN
      STA (ZPAGE),Y ;BUT MODIFIED FOR MOTOR X
      PLA
      DEX
      BPL EXCEPT
;++ TABLE IS NOW LOADED ++
      LDY #$30
WAIT
      DEY          ;FILL IN SOME
      BNE WAIT     ;TIME

      LDA #$FF     ;ALL PULSES ON
      LDY #$00
LOOP
      AND (ZPAGE),Y ;BUT MASK OFF WITH EACH
      STA PORT      ;TABLE ELEMENT IN TURN
      INY
      BNE LOOP
;
      LDX #$07
      LDA #$FF
CLEAR
      LDY ANGLE,X   ;CLEAR ALL EXCEPTIONS
      STA (ZPAGE),Y
      DEX
      BPL CLEAR
;++ ALL PULSES SHOULD NOW BE FINISHED ++
      PLA
      TAX
      PLA           ;RESTORE REGISTERS
      TAY
      PLA
      PLP
      JMP (OURVEC)  ;JUMP TO NORMAL
                    ;IRQ ROUTINE
;
;++++ SMOOTH SERVO MOVER ++++
;
LOOP2
     LDX #$00
     LDY #$00
LOOP1
     LDA NEWPOS,X ;NEW POSN FOR SERVO X
     CMP ANGLE,X  ;IS OLD POSN SAME?
     BEQ MOVED.   ;IF = THEN DO NOTHING
     BCS ADD      ;IF > THEN ADD
     DEC ANGLE,X  ;ELSE SUBTRACT
     JMP INCRX
ADD
     INC ANGLE,X
     JMP INCRX
MOVED
     INY
INCRX
     INX
     CPX #$04     ;ALL 4 SERVOS DONE
     BNE LOOP1    ;IF NOT NEXT SERVO
     JSR WAITER   ;SHORT PAUSE
     CPY #$04     ;IF Y=4 THEN ALL MOVED
     BNE LOOP2
     RTS
;
WAITER
     TXA
     PHA          ;SAVE X & Y REGS
     TYA
     PHA
     LDY #$FF
     LDX DELFAC   ;GET DELAY FACTOR
ENCORE
     DEY
     NOP          ;256*8 CLOCK CYCLES
     NOP          ;WAIT
     BNE ENCORE
     DEX          ;IF MORE DELAY THEN
     BNE ENCORE   ;REPEAT
     PLA
     TAY
     PLA
     TAX
     RTS
;
;++++ CHECK KEYBOARD ++++
;
KEYTAB *=*+16
RESULT *=*+1
GETIN=$FFE4
;
CHECK
     PHA
     TXA
     PHA
     TYA
     PHA
NOKEY
     JSR GETIN  ;GET A CHAR
     CMP #$00   ;NO CHAR?
     BEQ NOKEY
     LDX #$00
COMPAR
     CMP KEYTAB,X
     BEQ EXIT     ;IS CHAR IN TABLE?
     INX
     CPX #17
```

IAN McKINNELL

```
      BNE COMPAR
      LDX #$FF      ;SIGNAL NOT FOUND
EXIT
      STX RESULT    ;STORE KEYTAB OFFSET
      PLA
      TAY
      PLA
      TAX
      PLA
      RTS
```

Arm Sequence Programmer

```
10 REM ************************
20 REM ************************
30 REM **                      **
40 REM ** CBM ARM SEQUENCE **
50 REM **    PROGRAMMER    **
70 REM **                      **
80 REM ************************
90 REM ************************
95 :
96 DN=8:REM FOR CASS DN=1
97 IF A=0 THEN A=1:LOAD"ROBARM.HEX",DN,1
100 GOSUB 1000:REM SET UP
110 PRINT CL$
120 X=5:Y=6:GOSUB10000:PRINT"WOULD YOU LIKE TO :"
130 X=4:Y=10:GOSUB10000:PRINT"1....PROGRAM NEW ARM
 SEQUENCE"
140 X=4:Y=12:GOSUB10000:PRINT"2....ADD MOVES TO A
PROGRAM"
150 X=4:Y=14:GOSUB10000:PRINT"3....REPLAY A FILE"
160 X=4:Y=16:GOSUB10000:PRINT"4....LEAVE PROGRAM"
170 GET G$:IF G$="" THEN 170
180 IF G$="1" THEN C=0:LM=0:REM REST ARRAY PTRS
190 IF G$="1" OR G$="2" THEN GOSUB2000:GOSUB3000:G
OSUB4000:GOSUB5000
200 IF G$="3" THEN GOSUB6000:GOSUB4000
210 IF G$="4" THEN PRINTCL$:END
220 GOTO 110
500 REM **** KEY FUNCTIONS ****
540 DX=DX+1:RETURN
550 DX=DX-1:IF DX<0 THEN DX=0
555 RETURN
560 P=PEEK(NP)+DX:IF P<256THEN POKENP,P
565 RETURN
570 P=PEEK(NP)-DX:IF P>0THEN POKENP,P
575 RETURN
580 P=PEEK(NP+1)+DX:IF P<256THEN POKENP+1,P
585 RETURN
590 P=PEEK(NP+1)-DX:IF P>0THEN POKENP+1,P
595 RETURN
600 P=PEEK(NP+2)-DX:IF P>0THEN POKENP+2,P
605 RETURN
610 P=PEEK(NP+2)+DX:IF P<256THEN POKENP+2,P
615 RETURN
620 P=PEEK(NP+3)+3*DX:IF P<256THEN POKENP+3,P
625 RETURN
630 P=PEEK(NP+3)-3*DX:IF P<256THEN POKENP+3,P
635 RETURN
640 FOR I=0 TO 3:POKE NP+I,R%(C,I):NEXT I:RETURN
650 C=C+1:IF C>MC THEN C=0
655 RETURN
660 C=C-1:IF C<0 THEN C=MC
665 RETURN
670 FOR I=0 TO 3:R%(C,I)=PEEK(NP+I):NEXT:C=C+1:RET
URN
680 FOR I=0 TO 3:R%(C,I)=PEEK(NP+I):NEXT:C=C+1:RET
URN
999 :
1000 REM **** SET UP ****
1010 C=0:NS=3:REM NO. OF SERVOS-1
1020 LM=0:OC=0:DF=10:REM DELAY FACTOR
1030 MC=100:REM MAX COUNT
1040 DIM R%(MC,NS):REM KEY POSITIONS ARRAY
1050 CL$=CHR$(147):REM CLEAR SCREEN
1060 DW$="":FOR I=1 TO 25:DW$=DW$+CHR$(17):NEXT I:
REM CURSOR DOWN
1065 POKE 650,128:REM SET KEYS TO REPEAT
1070 DL=49152:NP=49416:REM DELAY AND POSITION ADDR
ESSES
1075 GS=49155:OFF=49170:REM WEDGE ON/OFF SYS ADDRE
SSES
1077 MOVE=49281:REM MOVE SERVOS SYS ADDRESSES
1078 KEYTAB=49343:RESULT=49359:REM KEYPRESS ADDRES
SES
1079 CHECK=49360:REM KEY CHECK SYS ADDRESS
1080 REM ** READ KEYPRESS ASC DATA **
1090 FOR I=0 TO 14:READ A:POKE KEYTAB+I,A:NEXT I
1100 DATA 73,69,157,29,145,17,65,90,88,67
1110 DATA 82,78,66,83,81
1200 REM ** ZERO NEWPOS LOCATIONS **
1210 FOR I=0 TO 3:POKENP+I,0:NEXT I
1990 RETURN
1999 :
2000 REM **** INFORM ****
2010 PRINTCL$
2020 X=5:Y=5:GOSUB10000:PRINT"PLEASE USE :"
2030 X=1:Y=7:GOSUB10000:PRINT"CURSOR KEYS...FOR L/
R AND 1ST ARM U/D"
2040 X=1:Y=9:GOSUB10000:PRINT"A & Z.....FOR 2N
D ARM U/D"
2050 X=1:Y=11:GOSUB10000:PRINT"X & C.....FOR G
RAB OPEN/CLOSE"
2060 X=1:Y=13:GOSUB10000:PRINT"S.....TO SA
VE A POSITION"
2070 X=1:Y=15:GOSUB10000:PRINT"Q.....TO RE
TURN TO MENU"
2080 X=1:Y=17:GOSUB10000:PRINT"R.....TO MO
VE TO SAVED POSITION"
2090 X=1:Y=19:GOSUB10000:PRINT"N & B.....NEXT
AND BACK COUNT"
2100 X=1:Y=21:GOSUB10000:PRINT"E.....SET N
EW COUNT"
2110 X=1:Y=23:GOSUB10000:PRINT"I & D.....TO IN
C AND DEC SPEED"
2120 X=2:Y=2:GOSUB10000:PRINT"COUNT=";C;" ";
2130 FOR I=0 TO 3:PRINTR%(C,I);" ";:NEXT:PRINT
2135 GET A$:IF A$="" THEN 2135
2140 RETURN
2999 :
3000 REM **** PROGRAM ARM ****
3005 GOSUB7000:REM BLANK SCREEN AND START WEDGE
3010 POKE DL,1:REM SET DELAY FACTOR TO 1
3020 DX=8:REM CHANGE RATE
3030 SYS CHECK:REM CHECK KEYPRESS
3035 R=PEEK(RESULT)+1:IF R>15 THEN 3030:REM NON-VA
LID KEY
3037 ON R GOSUB 540,550,560,570,580,590,600,610,62
0,630,640,650,660,670
3190 IF C>LM THEN LM=C-1:REM RECORD MAX COUNT SO F
AR
3220 OC=C
3260 SYS MOVE:REM M/C MOVE SERVOS
3270 IF R=15 OR C>MC THEN GOSUB8000:RETURN:REM WED
GE OFF AND QUIT
3280 GOTO 3030:REM REPEAT
3999 :
4000 REM **** REPLAY MOVEMENT SEQUENCE ****
4010 PRINTCL$
4020 X=5:Y=23:GOSUB10000:PRINT"REPLAY SEQUENCE (Y/
N), R REPEATS"
4030 GET AN$:IF AN$<>"Y" AND AN$<>"N" AND AN$<>"R"
 THEN 4030
4040 IF AN$="N" THEN RETURN
4050 IF AN$<>"R" THEN X=5:Y=22:GOSUB10000:INPUT"DE
LAY FACTOR 1-255";DF
4060 IF DF<0 OR DF>255 THEN 4050:REM CHECK RANGE
4070 POKE DL,DF:REM SET DELAY FACTOR REGISTER
4075 GOSUB7000:REM BLANK SCREEN AND START WEDGE
4080 FOR I=0 TO LM
4090 X=5:Y=2:GOSUB10000:PRINT"NO. IN SEQUENCE = ";
I;" ";
4100 FOR S=0 TO 3
4110 POKE NP+S,R%(I,S)
4120 NEXT S:SYS MOVE:REM M/C MOVE SERVOS
4130 NEXT I
4135 GOSUB8000:REM RESTORE SCREEN AND TURN OFF WED
GE
4140 GOTO 4010:REM REPEAT
4999 :
5000 REM **** SAVE A SEQUENCE ****
5010 PRINTCL$
5020 X=5:Y=10:GOSUB10000:PRINT"SAVE SEQUENCE ON FI
LE (Y/N)"
5030 GET AN$:IF AN$<>"Y" AND AN$<>"N" THEN 5030
5040 IF AN$="N" THEN RETURN
5050 X=5:Y=12:GOSUB10000:INPUT"FILENAME";FL$
5060 OPEN 3,DN,3,"@0:"+FL$+".DATA,S,W":REM OPEN FI
LE
5070 PRINT#3,LM
5080 FOR A=0 TO LM:FOR B=0 TO 3
5090 PRINT#3,R%(A,B):NEXT B,A
5100 PRINT#3:CLOSE3
5120 RETURN
5999 :
6000 REM **** LOAD A FILE ****
6010 PRINTCL$
6020 X=5:Y=10:GOSUB10000:INPUT"FILENAME TO LOAD";F
L$
6040 OPEN 3,DN,3,"@0:"+FL$+".DATA,S,R":REM OPEN FI
LE
6050 INPUT#3,C:LM=C
6060 FOR A=0 TO C:FOR B=0 TO 3
6070 INPUT#3,R%(A,B):NEXT B,A
6080 PRINT#3:CLOSE3
6100 RETURN
6999 :
7000 REM **** START WEDGE ****
7010 POKE53265,PEEK(53265)AND 239:REM BLANK SCREEN

7015 SYS GS
7020 RETURN
7999 :
8000 REM **** STOP WEDGE ****
8010 POKE53265,PEEK(53265)OR 16:REM BLANK SCREEN
8015 SYS OFF
8020 RETURN
8999 :
10000 REM **** POSITION CURSOR AT X,Y ****
10010 PRINTCHR$(19);:PRINTTAB(X);LEFT$(DW$,Y);
10020 RETURN
```



SIGHT EFFECTS

We turn our attention to the Commodore 64's video controller chip — the VIC-II — and look at how its registers can be used to produce different display modes and unusual visual effects. In the first of two parts, we consider low-resolution graphics, the ways that multicolour graphics can be produced and how the VIC-II chip organises its memory requirements.

There are eight basic screen graphics modes on the Commodore 64. In low-resolution graphics the character set may be in ROM or RAM and displayed in one of three possible modes: Standard mode, Multicolour mode or Extended colour mode. Also, there are two high-resolution modes: Standard and Multicolour. In addition to these basic modes, a number of other variations are possible: the screen can be set up to have 38 columns, as opposed to the normal 40 columns, and/or 24 rows, as opposed to 25. These last features are normally used in conjunction with smooth horizontal or vertical scrolling. The scrolling feature of the Commodore 64 is best run from machine code, since replacement display data needs to be shifted into the screen RAM fairly rapidly. On page 937 we designed a horizontal smooth-scrolling program that used the 38-column screen mode.

If a high-resolution screen is used in conjunction with a long program, then memory may be at a premium. The Commodore 64 allows considerable freedom in the location of both the low-resolution and high-resolution screen, so we will begin by looking at how the programmer can move them both around in memory. As we shall see in the next instalment, if you are using machine code to do the plotting, it is even possible to place the high-resolution screen *behind* the BASIC interpreter ROM (see page 1137). When you think about it, this is a great idea because if you are only running machine code, the interpreter ROM would be taking up eight Kbytes of valuable memory space doing nothing useful! By an interesting coincidence, eight Kbytes is just what we need for a high-resolution screen.

It is the function of the 6566/67 Video Interface Chip (VIC-II) to generate the video display data, which is passed to the television or monitor. To do this, the VIC-II must be able to read data from RAM or ROM. It is worth understanding the way in which the VIC-II obtains its data, since this gives rise to some of the more obscure behaviour of the Commodore 64.

Table 1

Bit Pattern Of Pixel Pair		Colour	Determined By
0	0	Screen background	53281 ($D021) Bits 0 to 4
0	1	Multicolour#1	53282 ($D022) Bits 0 to 4
1	0	Multicolour#2	53283 ($D023) Bits 0 to 4
1	1	Foreground colour	Bits 0 to 3 of colour nybble

Table 2

Screen Code	Bit 7	Bit 6	Background Colour Register
64 — 127	0	1	53282 ($D022) Extended colour 1
128 — 191	1	0	53283 ($D023) Extended colour 2
192 — 255	1	1	53284 ($D024) Extended colour 3

MULTICOLOUR MODE

Normal low-resolution display and bit map mode for the Commodore 64 have been described in earlier instalments (see pages 214 and 337). In this section we take a look at two of the other ways the machine's graphics can be used.

In Multicolour mode it is possible to have four colours within a single character cell, as opposed to two. There is a price to be paid, however, in that horizontal resolution is now in pairs of pixel dots rather than single pixels. Multicolour mode can be used in high- or low-resolution graphics, although the dot colours are determined slightly differently in the high resolution mode. The following POKEs from BASIC switch Multicolour mode on and off.

```
POKE 53270,PEEK(53270)OR16
POKE 53270,PEEK(53270)AND239
```

In low-resolution mode, when Multicolour is on, if bit 4 of the associated colour nybble (i.e. four bits, or half a byte) is one, then the character is interpreted in Multicolour mode, with the lower three bits determining the colour. This means that those characters that have associated colour nybbles in the range 0 to 7 are interpreted normally; whereas, if the colour code lies in the range 8 to 15, then the character will be displayed in Multicolour mode. The colours of the pixel pairs are determined as shown in Table 1.

By flipping the contents of addresses 53282 and 53283, we can instantaneously change the colour of every associated Multicolour pixel pair. It should be noted that Multicolour works best with user-defined characters — that is, the bit patterns take into account the fact that they are interpreted as pairs.

Another graphics mode available to the Commodore 64 programmer is Extended colour mode. This mode allows control over the background colour of the first 64 characters in the

character matrix. Extended colour mode cannot be used in conjunction with Multicolour mode. The following lines of BASIC turn Extended colour mode on and off, respectively:

```
POKE 53265, PEEK(53265)OR64
POKE 53265, PEEK(53265)AND191
```

The same character can appear on the screen with up to four different background colours, one of which is the screen background colour. The remaining characters cannot be displayed in Extended colour mode, since bits 6 and 7 of the screen code are used to control indirectly the colour of the character. For example, the screen code for 'A' is 1, and that of a reverse field 'A' is 65. However, if we POKEd 65 to the screen in Extended colour mode, we would not see a reverse field 'A' but a normal 'A' with a background colour determined by the contents of address 53282 ($D022). Similarly, POKEing 129 to the screen would produce a normal 'A', but with a background colour determined by the contents of address 53283 ($D023). Table 2 shows how the screen codes and colour registers are related.

MEMORY MANAGEMENT

The VIC-II chip sees a different and far simpler memory map from the 6510. At any one time, the VIC-II can see only one of four 16 Kbyte blocks — or banks — of memory. We can think of this 16 Kbyte bank as the VIC-II 'window' onto memory. The base address of this window can take one of four values under software control.

```
1000 REM**SELECT BANK FOR VIC WINDOW**
1010 POKE 56578,PEEK(56578)OR3:REM DDR
     CIA#2 BITS 0,1 TO OUTPUT
1020 WD=3:REM THIS SELECTS NORMAL
     WINDOW
1030 POKE 56576,(PEEK(56576)AND252)OR WD:
     REM PORT A CIA#1 BITS 0,1
```

Here WD can take values in the range 0 to 3. At any time, the value of WD currently in effect is found as PEEK(56576)AND3. The corresponding address of the bottom of the 16 Kbyte window in memory is computed using the formula: WB=16384*(3-WD), where the value in WD gives the corresponding addresses:

WD	Window Start Address
0	49152 ($C000)
1	32768 ($8000)
2	16384 ($4000)
3	0 ($0000)

Within the VIC-II window the 6510 expects to see the screen memory and a character ROM image in low resolution (or high-resolution data if high-resolution mode is selected). It may also need to find sprite data, and if so will need to see this within the window. The pointers for each of the eight sprites are placed at the end of the screen memory (and so must be moved with the screen).

The offset of the start of screen memory from the base of the current VIC-II window is controlled by the upper four bits of the VIC-II control register at address 53272 ($D018). Using these four bits, we can place the screen at any one of 16 Kbyte blocks within the window.

```
1040 REM**SELECT SCREEN OFFSET FROM
     BASE OF WINDOW**
1050 SO=1:REM THIS SELECTS NORMAL OFFSET
1060 POKE 53272,(PEEK(53272)AND15)OR 16*SO
```

Here SO can take values in the range 0 to 15. At any time the value of SO currently in effect is found using PEEK(53272)AND15. The corresponding address of the base of the current screen in memory is computed using either SC=WB+1024*SO (window base plus offset), or:

```
SC=16384*(3—(PEEK(56576)AND3))+64*(PEEK
(53272)AND240)
```

One problem in moving the screen about is that the colour RAM does not move. So, if we wish to have an alternate screen, we need a small machine code routine to swap the colour memory in and out of a buffer as appropriate. This was used as the basis for the alternate screens program for the Commodore 64, given on page 918.

A further factor to consider is that if you wish to PRINT to the new screen, then it is necessary to tell the *operating system* (as opposed to the video chip) where the new screen is located. This can be done by a POKE (or STA) to address 648 ($0288), which is a pointer that should contain the high byte of the screen memory base address. If SC is computed as above, the following BASIC code will do the job:

```
POKE 648,INT(SC/256)
```

To select the base address of the character matrix or high-resolution screen, we use the following code:

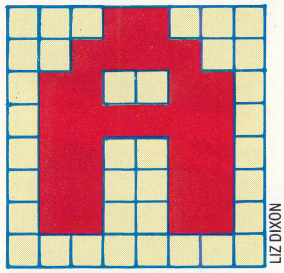
```
1070 REM**SELECT HIRES/CM OFFSET FROM
     BASE OF WINDOW**
1080 HO=4:REM THIS SELECTS NORMAL OFFSET
1090 POKE 53272, (PEEK(53272)AND240)OR
     2*HO
```

In principle, HO can take any value in the range 0 to 7, but in practice other factors limit the options. With WD equivalent to 1 or 3 we can't allow HO to have a value of 2 or 3, since this is where VIC-II sees the normal ROM character image. Also, large values of HO will put the top of the high-resolution memory out of reach of VIC-II. The corresponding address of the base of the current character matrix/high-resolution screen in memory is computed as: CM=WB+2048*HO (window base plus offset), or:

```
CM=16384*(3—(PEEK(56576)AND3))+1024*
(PEEK(53272)AND14)
```

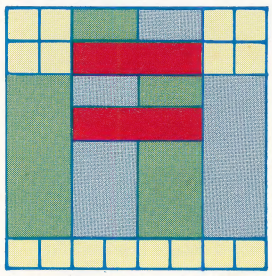
By setting these registers, we can move all the video display memory around in any way that suits the needs of our program.

Colour Co-ordinated
In the Commodore 64's normal display mode each bit set to one within the eight bytes that define a character is displayed in the current foreground colour. Any bits set to zero are displayed in the background screen colour. When multicolour mode is selected, the bit patterns are no longer interpreted as individual bits but as pairs. The four possible combinations of a pair of bits represent the foreground and background colours and two extra multicolours



'A' in normal display mode

```
0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0
```



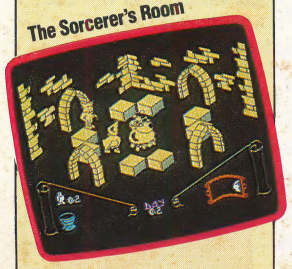
'A' in multicolour mode

```
0 0  0 1  1 0  0 0
0 0  1 1  1 1  0 0
0 1  1 0  0 1  1 0
0 1  1 1  1 1  1 0
0 1  1 0  0 1  1 0
0 1  1 0  0 1  1 0
0 1  1 0  0 1  1 0
0 0  0 0  0 0  0 0
```

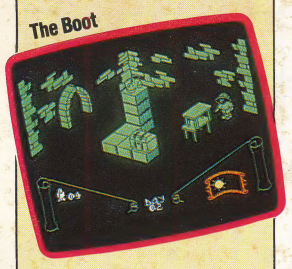
Key

- 01 MULTICOLOUR 1
- 10 MULTICOLOUR 2
- 11 FOREGROUND COLOUR
- 00 BACKGROUND COLOUR

BAD MOON RISING



The Sorcerer's Room



The Boot

The Cup

Lycanthrope's Lair
The aim of Knight Lore is to bring to the cauldron the object needed to prevent the knight from turning into a werewolf. To do this the knight must first visit the sorcerer to discover what is required. Then the quest can begin in earnest. Although the objects may change position, they are always found in the same rooms. For example, to get to the boot you must move the tables to the column and climb up. On the other hand, to get to the cup the player must wait for the right gate to open

IAN McKINNELL

On the heels of such popular favourites as Sabre Wulf and Atic Atac, Ultimate has now introduced a new and thoroughly different type of game: Knight Lore. With three-dimensional graphics and a complete control of Sabre Man's movements, this game is likely to become an even bigger success among Spectrum games players.

Ultimate has always been noted for producing quality games for the Sinclair Spectrum. The company has produced a long line of arcade-style maze games in which the player has to fight a way past strange and wonderful creatures that inhabit the various rooms. In the meantime, the player must also collect treasure, as well as pieces of talismans and other mystical trinkets, which when put together will complete the game. Examples of this type are Atic Atac (see page 376) and Sabre Wulf (see page 433).

Despite these games' popularity with Spectrum owners, the company has now moved away from the two-dimensional arcade adventure typified by Sabre Wulf. Although bearing some similarities with previous Ultimate creations, Knight Lore is a very different type of game. The scenario is that a knight (in fact, the hero of several previous adventures, Sabre Man) must search a wizard's castle in order to find the spell which prevents him from permanently becoming a werewolf. The player has forty days and nights to find the answer, the passing of which are displayed at the bottom of the screen; and in the bottom right-hand corner is a window containing a 'sundial'. During the day the player is Sabre Man. However, when the moon rises in the window, he is transformed, complete with convulsions in the best Lon Chaney manner, into the Werewulf. Luckily, this has no effect on the player's ability to move around except during the transformation itself, which can be a nuisance if you are trying to avoid an oncoming gargoyle.

On commencing the game, the first thing you will notice is the three-dimensional graphics that have been incorporated into the adventure. These graphics are the best yet to be seen on the Spectrum, including Ant Attack. Unlike Ant Attack, however, you are unable to change the viewing angle on Knight Lore, but this is compensated for by the ingenious ways in which the various rooms have been constructed.

Knight Lore is not an arcade adventure *per se*, but rather a series of logic puzzles that have to be solved. In this respect, Knight Lore resembles the true adventure game, posing tough problems that have to be dealt with before any progress can be made. Although many of the rooms are empty, others present you with challenges before you are able to get to the exit on the other side. One room, for example, is inhabited by small ghost-like creatures running around on the floor — if they touch Sabre Man you lose a life.

The puzzles have varying degrees of difficulty. Some are comparatively easy, others are fiendish in their complexity, and some are merely red herrings. You may come to a room with blocks in it, for example, above which are large globes with spikes on them. Attempt to cross the blocks and these spikey orbs will puncture your enthusiasm — the answer is to simply walk around the blocks! Another trick used by the programmers is to employ the distorted perspective one gets when seeing a two-dimensional representation of a three-dimensional space. Often, blocks that seem to be on the ground are in fact floating in the air, and should Sabre Man fall off one, there is no way out except by losing a life on the spikes and attempting to cross the room again.

In many of the rooms there are objects to collect such as gems, chalices and potions. Picking them up, once you've reached them, is accomplished by jumping on top of the object and pulling the joystick back. This will place the object in your 'inventory' that is displayed at the bottom left-hand corner of the screen. This manoeuvre could be useful to gain extra height when climbing over an otherwise unscalable wall.

Sabre Man can move in all four directions by using either the keyboard or joystick. Jumping is performed by pressing either the keys on the third row or the fire button. Correct orientation of Sabre Man is crucial to a successful game — jumping in the wrong direction, for example, could be fatal. You may well spend the first few games simply getting used to 'fine tuning' the correct orientation since very slight movements of the joystick can alter the way Sabre Man is facing.

Knight Lore is a fascinating game and well up to Ultimate's high standard. Even players who don't usually like encountering puzzles should find the game thoroughly absorbing, because the solutions to many of the problems require quick responses on the keys or fire button, as well as demanding a sharp and attentive mind.

Knight Lore: For the 48K Spectrum, £9.95
Publishers: Ashby Computers and Graphics Ltd, Ashby de la Zouch, Leicestershire, LE6 5JU
Authors: Ultimate Play the Game
Joysticks: Optional
Format: Cassette

THE HOME COMPUTER ADVANCED COURSE

INDEX TO ISSUES 49 TO 60

A

Acorn Electron 1069-1071
Acornsoft LOGO **1054**
ADA 987
ALGOL 60 987
ALGOL 68 987
AMASE 1042
AMPLE 1090
AMX mouse **1050-1051**
Apple Macintosh 984, 1023, 1125, 1181
Apricot portable **1009-1011**
Arrays 1146
Attributes 995

B

Background Music program 1118
Base 988
BaseValue program 1180
BBC BASIC **1037-1039**
BBC Micro
- *adventure game* 966-968, 997-999, 1017-1020
- *AMX mouse* 1050-1051
- *LOGO packages* 1054-1055
- *operating system* 978-980, 994-996, 1015-1016, 1037-1039, 1056-1057, 1077-1079, 1099-1100, 1117-1119
- *robot arm sequence program* 1177
- *simulation game* 1052-1053, 1066-1067, 1086-1088, 1106-1108, 1132-1133, 1144-1145, 1172-1174, 1192-1194

BCPL 987
Big Trak 1082
Bingo! game 1116
Bulletin boards **981-983**
Bumpers program 1036
Bus network 969

C

C 987
CAL 1003, 1141
CALL command 1056
Card Box 1105
CASE statement 1047
CBBS 981
CD-ROM 1182-1183
Cheung, Paul 1082
Christensen, Borge 1122
Circular 1084
COBOL 987
CODASYL database 1165
Cognitive skills 1022
Collector 988
COMAL 1122
Commodore communications modem 964
Commodore 64
- *adventure game* 966-968, 997-999, 1017-1020
- *BASIC pointer table* 1138
- *clock program* 1139-1140
- *memory map* 1137
- *operating systems* 1137-1140, 1157-1160, 1166-1167, 1198-1199
- *ports* 1157
- *robot arm sequence program* 1195-1197
- *simulation game* 1052-1053, 1066-1067, 1086-1088, 1106-1108, 1132-1133, 1144-1145, 1172-1174, 1192-1194
- *sound* 961-962
- *vector table* 1138

Communications **963-965, 981-983**
Compound statement 1025
Compunet 963-965
Concurrent PASCAL 987
Conditional loop 1008
Cray-1 1049
Critical path analysis 984

D

The Dark Wheel 1040
Data types 1147
Databases **1097-1098, 1101-1103, 1104-1105, 1124-1125, 1152-1153, 1164-1165, 1186-1187**
dBase II 1152, 1186
DBMs 1098
Dialog 1102
Digitaya game 968, 1018-1020
Discovery 1 **1154-1156**
Distances program 1136
Domesday Project 1182
Dragon 1064-1065

E

Econet 995
Education **1001-1003, 1021-1023, 1041-1043, 1061-1063, 1081-1083, 1101-1103, 1121-1123, 1141-1143, 1161-1163, 1181-1183**
Elite **1040**
Emitter 988
Enterprise Sixty Four **989-991**
EPROM chips 1044-1045
Equality 1025
Eratosthenes' Sieve 1148
Error handling 1038
Escher M C 972-973
Events 1099-1100

F

Factfile 1102
Farmfax **1064-1065**
Fields 1104
Filevision 1125
Fischertechnik robotics kit **1189-1191**
FOR statement 1093
FORTRAN 987

G

Ghostbusters **1000**
Golden program 1076
Graphics Movement program 1119

H

Handicapped users **1041-1043**
Haunted Forest game 966-967, 998-999
Hierarchical database 1164-1165
Homo Ludens 1063
Hybrid Music 500 **1090-1091**

I

IF statement 1046
ILECC 1161
Infra-red trackball 1011
Interrupt handling 1077-1079
Intrinsic programming 1002
ISO PASCAL 987

K

Keys 1124
Knight Lore **1200**

L

LAN 969
Language workspace 1038
LDR test program 1058
Light pen 969
Line following program 1060
Line printer 969
Line Tracer II 1084
Linear programming 1002
LISP 969
Local area network 969
Logging on 982, 988
Logic 969
Logic gate 988
Logic state 988
Logic symbols 988
LOGO **972-974,** 1054-1055 1082-1083, 1122
Logotron LOGO **1055**
Loop 1008
Low-level language 1008

M

Machine code 1008
Machine independent 1008
Macproject **984-985**
Macro 1028
Magnetic card 1028
Magnetic disk 1028
Mail box 1028
Mainframe 1049
Mark sensing 1049
Mass storage 1049
Matrix 1068

THE HOME COMPUTER ADVANCED COURSE

INDEX TO ISSUES 49 TO 60

MaxBreak program 1135
Memocon Crawler 1085
Memory hierarchy 1068
Micron EPROM programmer **1044-1045**
Micronet 800 963-965
Microprocessor 1068
Microspeech 970
Microsoft Project **1006-1007**
MIDI 1029, 1068
Mind At Play 1061
Minicomputer 1089
MIS 1049
MEP 1002
Modem 1089
Modula-2 987
Modular programming 1089
Modulation 1089
Modulus 1089
Monkey 1084
Monitor 1120
Mother board 1120
Mouse 1120
Movits **1084-1085**
Music Maker **961-962**
My Talking Computer **970-971**

N

Name 1128
NAND 1128
National Development Program 1002, 1162
Negative logic 1128
Nesting 1128
Network 1149
Network architecture 1149
Network database 1165
New World game 1053, 1067, 1088, 1108, 1133, 1145, 1174, 1194
Noise 1149
NOR 1149
NOT 1149
Numerical analysis 1149

O

Object code 1168
Object-oriented programming 1168
OCCAM 987
Octal notation 1168
One World 1103
One's complement 1168
Operand 1168
Operating systems **978-980, 994-996, 1015-1016, 1037-1039, 1056- 1057, 1077-1079, 1117-1119, 1137-1140, 1157-1160, 1166-1167,** 1168, **1198-1199**
Operation 1168
Operator precedence 1115
Optical disc 1188
Optimisation 1188
Opus 1154
OR 1188
Ordered pair 1188
OSARGS call 1015
OSBGET call 1015
OSBPUT call 1016
Oscilloscope 1188
OSFILE calls 994-996
OSFIND call 1015
OSGBPB call 1016
OSWORD calls 978-980

P

Packet Switching Service 964
Pan Course Tutors 1142
Papert, Seymour 1001-1003, 1082, 1163, 1182
Parameter blocks 978
Parameter passing 1057, 1178
Parawedge program 1167
Pascal, Blaise 986
PASCAL **986-987, 1004-1005, 1024- 1025, 1046-1048, 1074-1076, 1092-1093, 1114-1116, 1134- 1136, 1146-1148, 1178-1180, 1184-1185**
Penman Plotter **1169-1171**
Peripheral Ports 1157-1160
Perkins Brailler 1042
Photonic wand 1041
Piaget, Jean 1002
Piper Mouse 1084
PLATO 1003, 1161
PL/1 987
Plus 3 interface **1069-1071**
Prestel 963-965
PROLOG 1122
Public access systems 963-965

R

RealAle program 1093
Records 1134, 1152
Relational database 1164
REPEAT statement 1092
Reserved words 1005
Ring network 969
Robot arms 1072-1073
Robot controller program 1036
Rotronics Wafadrive **1129-1131**
RS232 1160

S

Sanyo MBC-550 **1109-1111**
Scope 1178-1179
SEMERCs 1041
Set operators 1135
Sets 1103, 1114
Shape scanner program 975-977
SID chip 961-962
SIGs 981-982
Simulations **1026-1027**
Simulation-Database Project 1062
Sinclair Spectrum
adventure game 966-968, 997- 999, 1017-1020
robot interface 992-993, 1012- 1014
robot control programs 1036
simulation game 1052-1053, 1066-1067, 1086-1088, 1106-1108, 1132-1133, 1144-1145, 1172-1174, 1192-1194
storage devices 1129-1131, 1154-1156
Skinner, B F 1001-1002
SMILE 1162
Speech recognition 1010
Speech synthesis 971
Star network 969
Stonier, Tom 1183
Subrange types 1076
Syntax diagrams 1004
Sysops 981-982

T

TBBS 981
Telecom Gold 963-965
Tessellations 972-974
"The Word" processor **1032-1033**
TICCIT 1003, 1161
Turtles 1081
Turtle geometry 973

U

UCSD PASCAL 987
Unconditional loop 1008
Use of memory program 1117
USR call 1056

V

Variable declaration 1024-1025
VIA chip 1079

W

Wafadrive **1129-1131**
Wedge code 1139, 1166
WHILE statement 1092
Wirth, Professor Niklaus 986
WITH statement 1135
Workshop robot **975-977, 992-993, 1012-1014, 1034-1036, 1058-1060**
Workshop robot arm **1072-1073, 1094-1096, 1112-1113, 1126- 1127, 1150-1151, 1175-1177, 1195-1197**

Y

Yamaha CX5M **1029-1031**

Z

Zaxxon **1080**